PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 :
G06F 15/00, 17/30

A1

(11) International Publication Number: WO 97/34240

(43) International Publication Date: 18 September 1997 (18.09.97)

(21) International Application Number: PCT/US97/04574

(22) International Filing Date: 17 March 1997 (17.03.97)

(30) Priority Data:

60/013,505	15 March 1996 (15.03.96)	US
08/792,371	3 February 1997 (03.02.97)	US

(71) Applicant: UNIVERSITY OF MASSACHUSETTS [US/US]; One University Avenue, Lowell, MA 01854 (US).

(72) Inventors: BUFORD, John, F.; 85 Butman Road, Lowell, MA 01852 (US). RUTLEDGE, John, L.; 54 Woodbury Lane, Acton, MA 01720 (US).

(74) Agent: LAND, John; Fish & Richardson P.C., Suite 1400, 4225 Executive Square, La Jolla, CA 92037 (US).

(81) Designated States: AU, CA, JP, European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

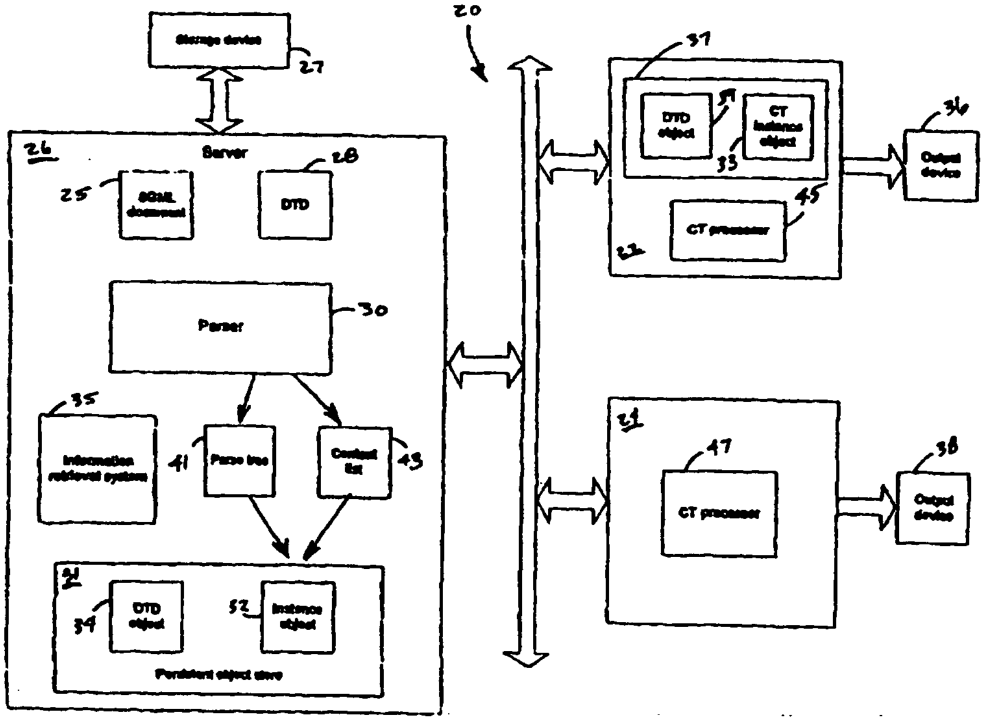
Published

With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: COMPACT TREE FOR STORAGE AND RETRIEVAL OF STRUCTURED HYPERMEDIA DOCUMENTS

(57) Abstract

A compact tree representation is used during the electronic storage, transmission, and presentation of a structured hypermedia document (25) in a networked computer. All text portions of the documents are pre-processed by a document parser (30) and the resulting document structure is stored in compact and compressed form in a persistent object storage (31) while the document content (32) is available in a compressed and indexable form consistent with full text retrieval systems. · During document delivery, the compact representation of the document (33, 39) is retrieved from the persistent object storage and transferred to a client computer (22), which reconstructs the document and presents it to a user (36). Any arbitrary structured document type can be stored and delivered this way. The CT representation can be partitioned dynamically or by preprocessing to allow the client to retrieve parts of the document incrementally, for non-linear or temporally ordered access. The CT representation can be used for group collaboration applications, including document sharing and document authoring applications.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AM	Armenia
AT	Austria
AU	Australia
BB	Barbados
BE	Belgium
BF	Burkina Faso
BG	Bulgaria
BJ	Benin
BR	Brazil
BY	Belarus
CA	Canada
CF	Central African Republic
CG	Congo
CH	Switzerland
CI	Côte d'Ivoire
CM	Cameroon
CN	China
CS	Czechoslovakia
CZ	Czech Republic
DE	Germany
DK	Denmark
EE	Estonia
ES	Spain
FI	Finland
FR	France
GA	Gabon
GB	United Kingdom
GE	Georgia
GN	Guinea
GR	Greece
HU	Hungary
IE	Ireland
IT	Italy
JP	Japan
KE	Kenya
KG	Kyrgystan
KP	Democratic People's Republic of Korea
KR	Republic of Korea
KZ	Kazakhstan
LI	Liechtenstein
LK	Sri Lanka
LR	Liberia
LT	Lithuania
LU	Luxembourg
LV	Latvia
MC	Monaco
MD	Republic of Moldova
MG	Madagascar
ML	Mali
MN	Mongolia
MR	Mauritania
MW	Malawi
MX	Mexico
NE	Niger
NL	Netherlands
NO	Norway
NZ	New Zealand
PL	Poland
PT	Portugal
RO	Romania
RU	Russian Federation
SD	Sudan
SE	Sweden
SG	Singapore
SI	Slovenia
SK	Slovakia
SN	Senegal
SZ	Swaziland
TD	Chad
TG	Togo
TJ	Tajikistan
TT	Trinidad and Tobago
UA	Ukraine
UG	Uganda
US	United States of America
UZ	Uzbekistan
VN	Viet Nam

COMPACT TREE FOR STORAGE AND RETRIEVAL OF STRUCTURED HYPERMEDIA DOCUMENTS

BACKGROUND OF THE INVENTION

1. *Field of the Invention*

This invention relates to electronic data storage, and more particularly to storage, retrieval, transmission, and presentation of structured hypermedia documents in distributed computing systems.

2. *Description of Related Art*

Structured documents are a class of electronic information storage in which the text content of a document includes embedded character sequences known as "markup" which identify structural elements and attributes or formatting codes for the content. The Standard Generalized Markup Language (SGML) is an example of a syntax for storing and processing structured documents (International Organization for Standardization, ISO International Standard 8879 – Standard Generalized Markup Language, 1985 Geneva, Switzerland). The HyTime standard is an example of a markup language for structured hypermedia documents (International Organization for Standardization, ISO International Standard 10744 – Hypermedia Time-Based Structuring Language (HyTime), 1992. Geneva, Switzerland). The Hypertext Markup Language (HTML), which is defined as an SGML document type definition (DTD), is a widely used markup language for hypertext documents (Berners-Lee, T., and Connolly, D., Hypertext Markup Language – 2.0, Internet Engineering Task Force RFC 1866, November 1995).

Under the SGML standard, a DTD defines the structural components that are required and allowed in a particular type of document. Each DTD begins with a declaration of the document type, *i.e.*, a statement that assigns an identifier to the particular document type (*e.g.*, a DTD defining a magazine may begin with the document type declaration "magazine"). All documents of the declared type should be structured according to the

DTD. The DTD then defines the elements, attributes, entities, and notations that may be used to compose a document of the given type. Elements are the components that create the logical structure of the document (*e.g.*, a magazine's elements are articles, which may consist of text, pictures, and graphical figures or tables). Attributes are the characteristics that each element type may take on in a document of the given type (*e.g.*, one attribute of an article is the page number on which it begins). Entities may be used to refer to long strings of text or to external files (*e.g.*, the term "Johnson article" may refer to an article in another magazine). Notations identify non-SGML components and provide instructions for using these components when presenting the document. Additional information on documents structured according to the SGML syntax may be found in *The SGML Primer*, SoftQuad, Inc., 1995.

One conventional approach by which structured documents in electronic form are delivered involves a client-server division. A client application executing on a local client processor retrieves a document in its native encoding from a server software program executing on a remote server processor (transfer). The client application then parses the document locally according to the grammar of the document structure (parsing), combines the output of the parser with local requirements regarding style attributes of individual elements (rendering), and presents the document and its content to the user interface (display). A retrieved document is transferred in bulk to the client.

Bandwidth is a scarce resource in present client-server systems, so it would be desirable to minimize the amount of data required to be transferred from a server to a client. Accordingly, a second approach by which structured documents are delivered is for the document server to transmit to the client a compressed form of the document's native encoding (compression), using conventional compression techniques such as those described in Witten, I. H., Moffat, A., and Bell, T., *Managing Gigabytes – Compressing and Indexing Documents and Images*, NY: Van Nostrand Reinhold 1994. The client retrieves the compressed document (transfer), decompresses the document to obtain the native encoding (decompression), and then parses the document locally according to the grammar of the document structure (parsing), combines the output of the parser with local requirements regarding style (rendering), and presents the document and its content to the user interface (display). However, a retrieved compressed document still is transferred in bulk to the client.

In these conventional approaches, a document authoring tool generates documents in a native encoding. Thereafter, the documents are transferred to the server and retained in the native encoding. If a document has been structured for time-dependent presentation (that is, some elements of the document are to be displayed before other elements), the internal scheduling information of the document is not used to schedule transmission of document elements from server to client. Instead, the document is transferred in bulk to the client for parsing, rendering, and time-dependent display at the client. Further, in the conventional approaches, multiple users accessing the same document for simultaneous collaboration each must retrieve a new copy of the entire document each time the original document is modified by another user.

Accordingly, it would be useful if the efficiency of the present client-server system of structured document access could be improved. The present invention provides a system and method that provides such improved efficiency.

SUMMARY OF THE INVENTION

The present invention provides a system and method of storage, retrieval, transmission, and presentation of structured hypermedia documents in client-server distributed computing systems. In the preferred embodiment, a network server maintains a persistent storage of an arbitrary number of preprocessed structured documents. Each structured document is processed and parsed once at the network server, and the result is stored in compact tree (CT) form in a persistent object store. The CT form of the document then is delivered to requesting client computers as one or more objects. When a document is edited at a client computer, the CT form or the edit operations can be transferred from the client to the server.

Since the document is retrieved in pre-parsed and pre-processed form, the client computer need not have a document parsing function. The parsing operation is done once at the server, and thereafter the resulting representation may be accessed many times by different clients. The clients do not need to parse the compact representation, because the representation retrieved by the client contains a parse tree and symbol definitions needed to render the document for presentation.

The compact tree techniques described above result in significantly smaller data transfers than traditional document transfer techniques. The actual bandwidth reductions depend upon the size and structure of the structured documents being transferred. Furthermore, transferring compact tree representations incrementally provides additional bandwidth and network performance improvements.

Since the CT representation and the original source representation of the document are essentially equivalent, the CT representation may replace the source representation, thereby saving storage space and cache memory space at the server. Similarly, since client computers generally cache documents in source format, cache memory usage in the client computers is reduced. The CT representation also preserves cache storage space at proxy servers that act as intermediate servers between the client computers and a remote server.

Since the CT representation of a structured document reduces the time needed to access the document, the network server is able to sustain more sessions in a given interval. Because client computers do not need to parse the document, the clients are able to present the document more quickly than when using traditional document transfer techniques.

Full-text and content-based retrieval are an important requirement for hypermedia document systems. The CT representation of structured documents separates the document's structural elements from its content, allowing the content to be stored and indexed by conventional full-text or content-based retrieval systems, while the structural portions can be queried by a structure-query processing language such as the HyQ query language [ISO 1992].

The CT representation also allows non-linear, partial-retrieval (incremental), and progressive access to documents. Since a document is stored on the server as a compact parse tree, client computers may access a subsection of the tree without retrieving and processing the entire tree. This speeds client access, particularly for large documents, since the compact tree can be partitioned into sub-trees deliverable in an arbitrary order, including non-linear and temporal ordering. Even though incremental access is useful in supporting non-linear and temporally-ordered access to portions of a document, conventional document transfer processes do not permit incremental, per document file access. Processing time also is reduced because the syntactical validation performed by the parser is performed only once at the server, rather than each time the document is accessed by a client.

In collaborative situations, multiple client computers may simultaneously view, and may even modify, the same document. To maintain document consistency among all clients during collaboration, changes made to the document at each client are propagated to the server and to the other clients. The object-oriented storage model of the document permits edit transaction objects to be incorporated into the incremental delivery of the document.

Advantages of the invention may include one or more of the following:

- A computer network may be simplified by eliminating the need for document parsers in client computers requesting access to structured documents.
- A structured document may be parsed once by a network server and then accessed indefinitely by multiple client computers.
- Network storage and bandwidth requirements may be reduced by transferring a compact representation of a structured document instead of transferring the entire document itself.
- Network performance may be improved by reducing the time required to access a structured document.
- Conventional information retrieval systems may be used to store and index the content of a structured document.
- Modifications to a structured document may be sent to a client user who is viewing the document without retransmitting the entire document.

The details of the preferred embodiment of the present invention are set forth in the accompanying drawings and the description below. Once the details of the invention are known, numerous additional advantages, innovations, and changes will become obvious to one skilled in the art.

BRIEF DESCRIPTION OF THE DRAWINGS

FIGURE 1 is a functional block diagram of a computer network that stores and transmits structured documents in a compact tree representation.

FIGURES 2A and 2B are flow diagrams of a network server's processing of a structured document.

FIGURE 3 is a flow diagram of a network client's processing of a compact tree representation of a structured document.

FIGURE 4 is a structural diagram of the elements of a document type definition (DTD) object for a compact tree representation of structured documents.

FIGURE 5 is a schematic diagram of the elements of an instance object for a compact tree representation of a structured document.

FIGURES 6A and 6B are block diagrams illustrating the storage and transmission format of a DTD object.

FIGURE 7 is a block diagram illustrating the storage and transmission format of the instance object.

FIGURES 8A and 8B are block diagrams of a computer network that stores and transmits structured documents in a partitioned compact tree representation.

FIGURES 9A and 9B are block diagrams of a computer network in which modifications to a structured document are transmitted between network computers.

FIGURES 10A and 10B are a document type definition (DTD) and an example SGML document generated according to the DTD, respectively.

FIGURES 11A and 11B are tables found in compact tree representations of the DTD and SGML document of FIGURES 10A and 10B.

FIGURE 12 is another example SGML document generated according to the DTD of FIGURE 10A.

FIGURES 13A and 13B are tables found in partitioned compact tree representations of the SGML document of FIGURE 12 and the corresponding DTD.

Like reference numbers and designations in the various drawings indicate like elements.

DETAILED DESCRIPTION OF THE INVENTION

Throughout this description, the preferred embodiment and examples shown should be considered as exemplars, rather than as limitations on the present invention.

Referring to FIGURE 1 and FIGURES 2A and 2B, in a first embodiment of the present invention, a structured document 25 encoded according to a standard syntax, such as SGML [ISO 1988], may be shared among the various computers of a computer network 20. In general, a network server computer 26 will receive a request for the document 25 from at least one of a set of client computers 22 and 24 (Step 50). Upon receiving the request, the server 26 retrieves the document 25, either from an associated storage device 27 if the server 26 is a computer system that permanently maintains the document 25 (*e.g.*, the server 26 is an Internet server), or from another computer system if the server does not permanently maintain the document (*e.g.*, the server 26 is a proxy server) (Step 52). The document 25 is accompanied by a DTD file 28 that defines the syntactical elements used to convert the document 25 into a compact tree (CT) representation. The DTD file 28 allows the document to be reconstructed from the CT representation and to be presented as formatted output.

After receiving the document 25, the server 26 uses an internal parser 30 to parse through the SGML document 25, breaking the document 25 into a parse tree 41 representing the document's structural hierarchy and a content list 43 representing the data contained in the document 25 (Step 54). The server 26 then converts the parse tree 41 and the content list 43 into structure and character data (CDATA) tables, respectively (Step 56). The structure tables store the document's structural information in compressed form, and the CDATA table stores the data content of the document 25 in compressed and indexable form. These tables are described in detail below. The server 26 then stores the structure and CDATA tables as a compact tree (CT) instance object 32 in a persistent data store 31, such as an object-oriented database, an object-oriented file system, or an object-relational database (Step 58). The instance object 32 is structured such that related structural elements (*i.e.*, the document's elements and their corresponding sub-elements) are located together in the compact tree, which allows the clients 22 and 24 and the server 26 to access the instance object, and therefore the structured document, incrementally, as described below. If the server 26 includes an information retrieval system 35 (Step 60), the server 26 passes the CDATA portion of the instance object 32 to the information retrieval system 35 for storage (Step 62). The server 26 also compresses the DTD file 28 into a compact DTD object 34 and stores it in the persistent object store 31 (Step 66), if the server 26 has not already done so while previously processing a document of the same type (Step 64). The server 26 uses standard compression techniques to create the instance object 32 and the DTD object 34.

Instead of delivering the entire SGML document 25 to the requesting client 22 or 24, the server 26 delivers only a copy 33 of the CT instance object 32 and, if necessary, a copy 39 of the CT DTD object 34. If the client 22 or 24 requests only a portion of the document (Step 68), the server 26 sends only the sub-objects of the CT instance object 32 relating to the requested portion (Step 70). Otherwise, the server 26 transmits the entire CT instance object to the requesting client 22 or 24 (Step 72). Furthermore, the server 26 transmits the DTD object to the client 22 or 24 (Step 76) only when the client specifically requests the DTD object 34 (Step 74). Since the DTD may be any SGML-conforming definition, the client generally will need to retrieve a copy 39 of the DTD object 34 with the document instance object 32. However, if the client 22 or 24 can process the document instance 32 without the DTD object 34, or if the client 22 or 24 already has a copy 39 of the DTD object 34, then only a copy 33 of the document instance object 32 need be transmitted to the client 22 or 24. The server 26 then waits for the next incremental or full-document request from the client 22 and 24 (Step 78).

If a document is structured according to a syntax that the server 26 does not recognize, the server 26 may follow one of several approaches. First, the server 26 may reject the document completely and notify the requesting client that the document could not be delivered. Second, if only a portion of the document is unrecognizable, the server 26 may ignore the unrecognizable portion and transmit the rest. Third, the parser 30 may replace the unrecognizable syntax with an acceptable syntax. Fourth, the server 26 may transmit the unrecognizable syntax to the client as character data (CDATA).

Referring also to FIGURE 3, after requesting the document or a portion thereof from the server 26, the client 22 or 24 receives a copy 33 of the CT instance object 32 or of the appropriate sub-objects (Step 82) and stores it in a local storage area 37, such as cache memory or a hard disk drive (Step 84). If the client 22 or 24 has not already received the DTD object 34 in response to an earlier request (Step 86), that client requests the DTD object 34 (Step 88) and, after receiving it, stores the object in the local storage area 37 (Step 90). Using the DTD object as a guide, a CT processor 45 or 47 in the client 22 or 24 then traverses the CT instance object (Step 92) and converts that object into a formatted document (Step 94). The client 22 or 24 then presents the formatted document to an output device 36 or 38, such as a video display or a printer (Step 96).

In a second embodiment of the present invention, documents are pre-processed in the server 26 so that they are available in processed form before requested by a client. In a third embodiment of the present invention, as documents are requested by clients and processed in the server 26 as above, the processed form is stored in the server so that they are available in processed form for future requests by a client.

Referring to FIGURE 4, a CT DTD object 34 preferably is organized into several tables, each of which contains a portion of the DTD information needed for recreation (rendering) and presentation of the structured document. In the preferred embodiment, the tables in the DTD object 34 include a generic identifier (GI) table 102, a name table 104, an attribute declaration table 106, an attribute enumeration table 108, a notation table 110, an entity table 112, and a character data (CDATA) table 114. Preferably, DTD information that is needed only at parse time (*e.g.*, the content model of each generic identifier) is discarded and is not placed into the DTD object 34. The name table 104 and the CDATA table 114 preferably consist of text strings; the other tables preferably consist of constant length numerical fields. In the preferred embodiment, the name table 104, notation table 110, entity table 112, and CDATA table 114 are "common" tables that may be shared by the DTD object 34 and the instance object 32. The GI table 102, attribute declaration table 106, and attribute enumeration table 108 are DTD-specific tables.

To simplify access to the information stored in the DTD object 34, each table preferably has constant-size entries which may be identified by simple numeric indices. Client computers can find information in a table by multiplying the entry index for the desired information by the predetermined size of the entries. Variable length information, such as a list of sub-elements (or "children" elements) associated with an element, preferably is stored as consecutive entries in either the name table 104 or the CDATA table 114. As a result, all variable length information may be represented by two numeric values: one value representing the index of the first table entry containing a portion of the desired information, and the other value representing the number of entries required to store the information. Storing variable length information in constant size table entries trades space efficiency for access efficiency. One of ordinary skill in the art would recognize that the tables may be designed differently to strike a different balance between space efficiency and access efficiency.

In the preferred embodiment, the GI table 102 maintains a list of generic identifiers associated with the attributes contained in the attribute declaration table 106. The GI table 102 preferably contains at least four fields for each generic identifier. The first field 120 indicates the index of the generic identifier string stored in the name table 104. The second field 122 indicates the index of the first attribute stored in the attribute declaration table 106 for each generic identifier. As described below, all attributes associated with a generic identifier are stored consecutively in the attribute declaration table 106, with fixed attributes followed by non-fixed attributes. The third field 124 and fourth field 126 of the GI table 102 indicate the number of fixed attributes and the number of non-fixed attributes, respectively, associated with each generic identifier. If desired, other information may be stored in the GI table 102.

The name table 104 preferably stores a single copy of the text string associated with each generic identifier. Each string contained in the table 104 is represented by an index that identifies the position of the string in the table 104. The indices are represented by a binary value containing the least number of bits required to identify every generic identifier (*i.e.*, $\log_2 N$, where N is the number of strings in the name table 104). By storing all identifier strings in the name table, the DTD object 34 reduces the storage space required to represent all string identifiers.

The attribute declaration table 106 maintains a list of the attributes associated with each generic identifier, their types, and their default values. The table 106 preferably includes at least three fields: a name field 128, a type field 130, and a value field 132. The name field 128 indicates the index of the attribute name stored in the name table 104. The type field 130 indicates whether the attribute is an identifier (ID), an IDREF, a name, an entity reference, an enumeration, or CDATA. If the attribute is an identifier (ID or IDREF), the value field 132 stores the index of the identifier string in the name table 104. If the attribute is an enumeration, the value field 132 indicates the index of a list of possible enumeration values stored in the attribute enumeration table 108. If the attribute is CDATA, the value field 132 indicates the index of the default value in the CDATA table 114. Because this information is stored in the attribute declaration table 106, the structured document and the CT instance object may contain a list of attributes that differ from their default values. The attribute declaration table 106 also eliminates the need to store information about fixed attributes in the document instance object.

The attribute enumeration table 108 maintains a list of possible enumeration values for each attribute declared in the attribute declaration table 106. The attribute enumeration table preferably contains at least two fields. An NameTable Entry field 133 stores the index of a first enumeration value in the name table 104. A No. of Entries field 134 stores the number of possible enumeration values. These enumeration values are stored consecutively in the name table 104, starting at the index stored in the NameTable Entry field 133. Optionally, an index field 135 is also provided to reference multiple entries in the attribute enumeration table 108. (Indeed, throughout these examples, all of the index columns are shown for clarity but are not explicitly part of the CT representation).

The notation table 110 preferably includes at least three fields: a name field 140, a type field 142, and a value field 144. The name field 140 stores an index to the notation identifier string stored in the name table 104 for each notation. The type field 142 contains a flag (preferably a single bit) for each notation that indicates whether the notation is a system or public notation. The value field stores an index to the corresponding notation value in the CDATA table 114.

The entity table 112 preferably consists of at least four fields: a name field 146, a type field 148, a notation field 150, and a data field 152. The name field 146 stores an index to the entity identifier string stored in the name table 104 for each entity. The type field 148 indicates whether the entity is notation data (NDATA), character data (CDATA), or specific character data (SDATA), each of which is known in the art. If the entity type is NDATA, the notation field 150 stores an index to the corresponding notation string in the notation table 110; otherwise, the notation field 150 is empty (or ignored). The data field 152 stores an index to a corresponding entity definition in the CDATA table 114.

The CDATA table 114 preferably stores unparsed character data, such as notation values, external entity values, and attribute values. Like the name table, the CDATA table 114 stores each variable length string in consecutive constant-length entries that may be indexed easily. After the DTD object 34 is fully constructed, the CDATA table 114 is compressed using a conventional text compression technique.

Referring to FIGURE 5, a CT instance object 32 also is composed of several tables, each of which contains a portion of the information required to reconstruct the document instance. In the preferred embodiment, the tables specific to the instance object 32 include at least an instance table 160, an attribute value table 162, and a child table 164, each of which is described below. These tables describe the document-specific element hierarchy and identify any attributes that are not set to the corresponding default values. The instance object 32 also may use the four common tables in the DTD object (*i.e.*, CDATA, name, notation, and entity). The instance object 32 preferably will include an entity table (*i.e.*, will use the common entity table) only when new external entities are used in the document instance. Likewise, the instance object 32 will include a notation table, GI table, or CDATA table only when new notations, identifiers, or CDATA, respectively, are used in the document instance. The index values for the instance object elements in the common tables are determined by treating each table as a continuation of the corresponding table in the DTD object.

The instance table 160 indicates the elements occurring in the document instance. The instance table 160 preferably includes at least five information fields for each element: a generic identifier (GI) field 166, a flag field 168, a first attribute field 170, a first child field 172, and a number-of-children field 174. The GI field 166 stores an index to a corresponding generic identifier in the GI table 102 of the DTD object. The flag field 168 contains a binary bit vector in which each bit represents a non-fixed attribute in the corresponding element. If the bit corresponding to a particular attribute is set, that attribute does not have the default value, but rather its value is determined by the value found in the first attribute field 170. The first attribute field 170 stores the index of the first non-default attribute value stored in the attribute value table 162 for the corresponding element. The attributes for which the corresponding bits in the flag field 168 are set are stored consecutively in the attribute value table 162, beginning at the index stored in the first attribute field 170. All attributes which are set to the corresponding default values are represented by cleared bits in the flag field 168 and are omitted in the attribute value table 162. The bits in the flag field bit vector and the attribute values in the attribute value table 162 appear in the same order that the non-fixed attributes for the corresponding generic identifier appear in the attribute declaration table 106. The first child field 172 stores the index of the corresponding element's first child element (or sub-element) in the child table 164. The number-of-children field 174 indicates how many children sub-elements the corresponding element has. All sub-elements of an element are stored consecutively and in proper hierarchical order in the child table 164.

The attribute value table 162 stores at least a value for every attribute that is not set to the corresponding default value. Each entry in this table consists of one value, an index to the identifier string in the name table 104 representing the default attribute value. The attribute name and attribute type need not be stored in this table because this information is stored in the attribute declaration table in the DTD object.

The child table 164 preserves the hierarchical structure of the elements occurring in the document instance. Each entry in the child table 164 is represented by at least two information fields: a child field 176 containing a single bit indicating whether the entry is itself an element or is a pseudo element, and a value field 178 storing an index to the instance table 160 if the entry is an element or an index to the CDATA table 114 if the entry is a pseudo element. A child element contains everything located between two start and end tags found within the parent element, while the corresponding pseudo elements contain the content data located before the start tag and after the end tag, respectively.

Each field in the tables of the instance object 32 preferably contains the smallest number of bits possible to present the required information. For example, the GI field 166 of the instance table 160 should contain only enough binary bits to represent the highest index in the GI table 102 of the DTD object. Similarly, the attribute value table 162 and the child table 164 are large enough only to index the respective tables in the DTD object. In the preferred embodiment, the bit vectors in the flag field 168 of the instance table 160 must include one bit for each non-fixed attribute in the element type having the most non-fixed attributes. The number-of-children field 174 must be large enough only to indicate the number of sub-elements in the element having the most sub-elements. Each child table entry must have enough bits to store the highest index value for the CDATA table 114 and the instance table 160, plus an additional bit to indicate the child type. Each entry in the attribute value table 162 must be large enough to store the highest index in the name table 104, CDATA table 114, entity table 112, or attribute enumeration table 108, whichever is larger. The entries in the attribute value table 162 may have more bits than the value field 132 of the attribute declaration table 106 because the document instance may have added additional information to the name table 104 and the CDATA table 114.

The last entry 180 in the instance table 160 is unique in that it provides special information required by client computers to recreate the document instance. This entry preferably has at least one sub-element. Additional elements are pseudo elements containing any SGML processing instructions defined in the DTD. SGML processing instructions, which are known in the art, provide system specific information that must be used to reconstruct the document instance and therefore must be included in the instance object. The SGML processing instructions are followed by the first child that is a true element (as opposed to a pseudo element), which indicates the top level element of the document. Any remaining children of the last instance table entry 180 may be used by the server to pass optional parameters, such as PI entity definitions, that help the server and the client optimize or customize delivery of the document instance.

The entries in the document instance table 160 are ordered according to a recursive, depth-first search of the parse tree. As a result, entries representing components of the same element occupy consecutive locations in the name table 104, entity table 112, CDATA table 114, instance table 160, attribute value table 162, and child table 164. Also, the entries representing a child element preferably appear before the entries of subsequent children of the same parent element. Furthermore, the entries representing an element's children preferably immediately precede the element's own entries in the document instance table 160.

Referring to FIGURES 6A and 6B, the server stores and transmits the DTD object in a format understood by the client computers. The preferred format of the DTD-specific portion 200 of the DTD object is shown in FIGURE 6A. The server first stores a fixed length integer field 202 representing the number of enumerations contained in the attribute enumeration table 108 (FIGURE 4). The server then stores a bit-packed version of the attribute enumeration table 108 in a variable length field 206. Following the attribute enumeration information is a fixed length integer field 208 indicating the number of entries in the attribute declaration table 106 (FIGURE 4) and a variable length field 210 containing a bit-packed version of the attribute declaration table 106. The server then stores a fixed length field 212 indicating the number of entries in the generic identifier (GI) table 102 (FIGURE 4), followed by a variable length field 214 containing a bit-packed version of the GI table 102.

The preferred format of the common portion 220 of the DTD object (*i.e.*, the portion that may contain document instance information as well as DTD information) is shown in FIGURE 6B. The first two fields are a fixed length integer field 222 indicating the number of bytes in the compressed CDATA table 114 (FIGURE 4) and a variable length field 224 containing the compressed CDATA table 114. The next two fields are a fixed length integer field 226 indicating the number of entries in the name table 104 (FIGURE 4) and a variable length field 228 containing the strings of the name table 104. These fields are followed by a fixed length integer field 230 indicating the number of entries in the notation table 110 (FIGURE 4) and a variable length field 232 containing a bit-packed version of the notation table 110. The last two fields of the common portion 220 of the DTD object are a fixed length integer field 234 indicating the number of entries in the entity table 112 (FIGURE 4) and a variable length field 236 containing a bit-packed version of the entity table 112.

Referring to FIGURE 7, the server also stores and transmits the instance object 32 in a format understood by the client computers. The server preferably first stores a fixed length integer field 242 indicating the number of entries in the attribute value table 162 (FIGURE 5), followed by a variable length field 244 containing a bit-packed version of the attribute value table 162. The next two fields include a fixed length integer field 246 indicating the number of entries in the child table 164 (FIGURE 5) and a fixed length integer field 248 indicating the number of entries in the instance table 160 (FIGURE 5).

These fixed length fields are followed by two variable length fields, the first field 250 containing a bit-packed version of the child table 164, and the second field 252 containing a bit-packed version of the instance table 160. As discussed above, the last entry in the instance table 160 identifies the DTD object corresponding to the instance object 32 and provides instructions for reconstructing the document instance. The client computers automatically retrieve the last entry in the instance table 160 to begin the reconstruction process.

Referring to FIGURES 8A and 8B, the server 26 may create a CT instance object 32 in a manner that allows incremental delivery of the instance object 32 in either linear or non-linear order. Incremental delivery permits a client to retrieve and present portions of a document that are needed immediately for viewing regardless of where they occur in the document. For example, the client may need to begin its presentation of a document with a hyperlink that occurs in the middle of the document. Incremental delivery also permits a progressive-style display of a document, such as displaying all of the document's main headings before displaying its subheadings and body. Incremental access also allows the client to retrieve additional portions of a document only as the user attempts to view them, so that the server must transfer only those portions of the document that are needed by the user. This provides more optimal use of system resources when, for example, the user hyperlinks to a new document before entirely viewing the current document. Incremental delivery also allows the client to access portions of a temporally organized document, so that parts of a document can be retrieved in a specific time order. Incremental transfer is important for hypertext access in which the user is likely to browse through documents without viewing their entire contents.

As shown in FIGURES 8A and 8B, the server 26 creates an instance object 32 ready for incremental delivery by effectively dividing the associated compact tree into sub-trees. The server 26 divides the compact tree by partitioning the tables in the CT instance object 32. Because all of the information about an element and its children is stored as consecutive entries in the instance object tables, the instance table may be partitioned easily. To do so, the parser 30 generates an index 260 of element boundaries that indicates the first and last entries in each instance object table containing information about each element and its corresponding sub-elements. The index 260 of element boundaries preferably consists of at least three sub-indices: a first sub-index 262 that indicates the index bounds of information stored in the attribute value table 162 (FIGURE 5) for each element and its corresponding sub-elements; a second sub-index 264 that indicates the index bounds of child information stored in the child table 164 (FIGURE 5) for each element; and a third sub-index 266 that indicates the index bounds of entries in the document instance table 160 (FIGURE 5) for each element and its sub-elements. Alternatively, the server 26 may divide the document into a uniform set of equally-sized partitions without regard to corresponding elements and sub-elements. For a temporally scheduled document, the server may partition the document instance object 32 into sub-trees that preserve the time order in which the portions of the document must be presented to a client.

When a client requests incremental delivery of a document, a document delivery engine 49 in the server 26 uses the index 260 of element boundaries to determine which entries from the three instance object tables (attribute value, child, and instance) must be sent to the requesting client. Instead of sending each of the three tables in its entirety along with a fixed length integer indicating the size of the table, the document delivery engine 49 delivers the requested range of entries along with a pair of fixed length integers, one of which indicates the index of the first entry in the delivered table fragment, and the other of which indicates the total number of entries in the delivered table fragment. When the first sub-tree of the instance object is sent to the client, a third integer is transmitted by the server 26 to indicate the total size of the corresponding table, which allows the client tc reserve enough memory space to receive all remaining fragments of the three tables, if necessary. The first two integers (*i.e.*, index of the first entry and the number of entries in the table fragment) are used by the client to place the corresponding table fragment in the proper position in the client's copies of the tables.

In general, the name table 104, notation table 110, and entity table 112 (*i.e.*, the common tables which may be shared by the DTD and instance objects) cannot be partitioned because the entries in these tables may be shared among sub-trees. However, the invention does not exclude the possibility that the CDATA table can be partitioned. Hence, there is the possibility that during increment transfer mode, the client computer may not receive the first CDATA partition that contains the desired DTD name (*i.e.*, the client may get some other partition first if, in some cases, the CDATA table is being partitioned). To deal with this case, a convention is used such that, during incremental transfer mode, the server will send the DTD name to the client right before the table boundaries are sent; this only needs to be done for the first partition.

When a client requests incremental delivery of an instance object that shares these tables with the DTD object, the server 26 may select between two alternative approaches. In the preferred embodiment, a single-bit flag stored in a flag register 270 is associated with each table to indicate which of the two approaches the server should use. If the flag bit is set, the table is small enough that the server 26 may send the entire table with each sub-tree. If the flag bit is cleared, the table is too large to be sent with each sub-tree, so the server 26 must select only those table entries that are needed to decode the particular sub-tree. In the latter situation, the server 26 transmits the numeric index of each entry selected from the table. The server 26 sets or clears the flag bits based upon the values of the integers associated with each table indicating the table size.

When the common tables are too large to send with each sub-tree, the server 26 may determine which entries to send to the client in one of two ways. First, the server 26, when it parses a document instance 25, may create and store a record indicating which entries in the common tables correspond to each partition in the DTD object 34. Second, when the server 26 creates a partition to deliver to the client, the server 26 may parse the portion of the original document 25 corresponding to the partition to determine which elements of the name table 104, notation table 110, and entity table are used in the partition. One of ordinary skill will recognize that the first alternative is preferred in a network that is more sensitive to increased processing overhead in the server during document delivery, and that the second alternative is preferred in a network that is more sensitive to increased storage overhead in the server.

Unlike the other three common tables, the CDATA table 114 can be partitioned since each piece of content data belongs to a unique element of the CT instance object 32. To support incremental delivery of documents, the server 26 independently compresses each entry in the CDATA table 114 and separately delivers each entry with the corresponding instance object sub-tree when incremental access is requested. This division of the information in the CDATA table should correspond to the partitioning of the document into sub-trees for depth first ordering.

To further improve incremental retrieval, the top elements of the document tree may be placed in a separate partition to which other sub-trees are attached. In large documents, the lower-level sub-trees also may be divided in a similar fashion. Sub-tree division is best performed in a DTD-specific way in order to use knowledge of the document structure for optimization.

In response to a client's request for incremental delivery of a document, the server transfers all of the entries for the highest-level sub-tree. The client then processes the high level structure of the document and requests only the sub-trees it requires. The client may use conventional addressing mechanisms such as SGML IDREF or HyTime location addressing forms to identify any element instance within a document. When the client returns a request for a specific element, the server uses the address of the requested element to create an instance object sub-tree containing information for the requested element and its corresponding sub-elements.

Progressive views of the document can be provided to the client by passing consecutive partitions of the same root sub-tree incrementally, but with successively increasing depth.

Alternatively, a root sub-tree having N levels can be delivered as a partition to the client, and the client can present the sub-tree progressively to the N levels.

Referring again to FIGURE 1, the server 26 may be a proxy server acting as an intermediate server between a conventional server and its client computers, including clients located behind a "firewall" and mobile clients. The proxy server may serve as a bridge to improve system security or system performance. If a remote server passes a native encoding document 25 to the proxy sever, the parser 30 in the proxy server will convert the document 25 into the compact tree (CT) format and then forward the CT document to the client.

Referring to FIGURES 9A and 9B, client computers 302 and 304 may engage in simultaneous collaborative author-mode access to a document stored as a CT instance object 306. At the same time, each client 302 and 304 may hold similar copies 308 and 310 of the CT instance object 306 for simultaneous viewing and modification. When one client 302 modifies its copy 308 of the instance object by deleting, inserting, appending, or replacing information, a document editor 312 in the client computer 302 creates an edit object 320 that stores the modifications. The edit object 320 preferably includes one fixed length integer field 322 containing a time-stamp for the modifications and another fixed length integer field 324 indicating the address of the instance object sub-tree affected by the modifications. The edit object 320 also preferably includes a table 326 having at least two fields, the first 328 of which indicates the index of each element in the instance object that was modified, and the second 330 of which identifies the modification that was made.

The modifying client 302 sends the edit object 320 to the server 300, which in turn sends the edit object 320 to the other client 304. A document editor 314 in the server 300 then uses the edit object to modify the instance object 306 accordingly. Likewise, a document editor 316 in the other client 304 uses the edit object 320 to modify its copy 310 of the instance object accordingly. Edit objects may be used with incremental delivery.

Example

FIGURES 10A and 10B show a document type definition (DTD) 352 and an SGML document 350 generated according to the DTD 352, respectively. The DTD 352 defines a slide show presentation with time dependent information. According to the DTD 352, each slide show presentation is made up of one or more slide elements 354. Each slide 354 in turn consists of one or more items 356 (or child elements), each item being either an image element 358 or an audio element 360. Each slide 354 also has two attributes: a "name" attribute 362 that uniquely identifies the corresponding slide, and a "next" attribute 364 that indicates the name of the next slide in the presentation.

Each item element 356 (image or audio) has two attributes providing temporal information about the presentation of the element. A "start" attribute 366 indicates an amount of time to delay presentation of the item after the corresponding slide presentation has begun. When the "start" attribute 366 is set to the default value 368 of zero, the image or audio element is presented precisely when the presentation of the slide begins. A "units" attribute 370 indicates whether the "start" attribute 366 is measured in seconds 372, minutes 374, or hours 376. According to the DTD 352, "seconds" is the default value 378 of the "units" attribute 370.

Image elements 358 and audio elements 360 each include a data attribute 380 and 381, respectively, that is an SGML external entity, such as an image file or an audio file. The DTD 352 defines a notation 382 for image elements ("gif" represents the ".gif" format for image data) and a notation 384 for audio elements ("ulaw" represents the ".ulaw" format for audio data). Both notations 382 and 384 are defined as "SYSTEM" type notations. Image elements 358 also include an "x" attribute 386 and a "y" attribute 388, which define the x-y location of the image in the slide presentation space. Both the "x" and "y" attributes 386 and 388 have default values 390 and 391 of zero.

In FIGURE 10B, the document instance 350 declares that the "slideshow" document is defined by the document type definition "slideshow.dtd" 464, shown in FIGURE 10A.

The document instance 350 also declares a notation and five entities not declared in the DTD 352. The declared notation 464 ("postscript") is assigned the Adobe PostScript 3.0 format 466 ("PS-Adobe-3.0"). The first two entities, an "audio-1" entity 468 and an "audio-2" entity 470, are "NDATA" entities associated with two audio data files, and "audio-1.ulaw" file 472 and an "audio-2.ulaw" file 474, respectively. The other three entities, an "image-1" entity 476, an "image-2" entity 478, and an "image-3" entity 480, are "NDATA" entities associated with three image data files, an "image-1.gif" file 482, "doc.ps" 486, and an "end.gif" file 484, respectively.

After the document type and entity declarations, the document instance 350 defines the highest-level element in the document hierarchy, the "slideshow" element 488, which has two sub-elements, a first "slide" 490 and a second "slide" 492. The first "slide" element 490 has two attributes, a "name" attribute 494 that indicates the name of the first "slide" element 490 ("first") and a "next" attribute 496 that indicates the name of the second slide element. Because no slide follows the second "slide" element 492, the second "slide" element 492 has only a "name" attribute 498 indicating the name of the second slide 492 ("end").

The "first" slide 490 consists of four items elements, two audio elements 500 and 506, and two image elements 502 and 504. The first audio and image elements 500 and 502, which include the "audio-1" and "image-1" entities, are displayed simultaneously with the beginning of the first slide 490 since no delay period is specified. The second image element 504, which includes of the "image-2" entity, is displayed two seconds after the first image element 502 is displayed (*i.e.,* the "start" attribute 507 for the second image 504 has a value of "2", and the "units" attribute 508 has a value of "seconds"). The second audio element 506, which includes of the "audio-2" entity, begins 130 seconds after the beginning of the "first" slide 490 (*i.e.,* the "start" attribute 510 for the second audio element has a value of "130").

The "end" slide 492, which is presented after the "first" slide 490, consists of a single image element 512. This image element 512 includes the "image-3" entity and is displayed simultaneously with the beginning of the "end" slide presentation.

FIGURES 11A and 11B show the tables that make up the compact trees (CT) for the DTD 352 and document instance 350. The CDATA table 392, the name table 394, and the notation table 396 are common tables containing data representing both the DTD 352 and the document instance 350. The other tables shown in FIGURE 11A contain data representing only the DTD 352, and the tables in FIGURE 11B contain data representing only the document instance 350.

The name table 394 of FIGURE 11A includes 18 entries corresponding to the DTD 352 of FIGURE 10A and 8 entries corresponding to the document instance 350 of FIGURE 10B. The first entry (index 0) in the name table 394 is a null string, which is associated with objects that do not have a name (*e.g.*, the last entry in the document instance table). The next 17 entries are entered according to a depth-first recursive pass through the DTD 352. Therefore, elements appearing at lower levels of the DTD hierarchy appear first in the name table 394, while elements appearing at higher levels of the DTD hierarchy appear later in the name table 394. For example, the "audio" element 360 and the "image" element 358 are the lowest level elements in the DTD 352, so the corresponding entries in the name table 394 (index 1 and index 3, respectively) appear before the entries representing "item" element 356 (index 6), "slide" element 354 (index 12), and "slideshow" element 353 (index 15). Likewise, the attributes associated with each element immediately follow the element name in the name table 394, unless a similarly named attribute appears higher in the table. For example, the entry representing the "data" attribute 381 of the "audio" element 360 immediately follows the name table entry for the "audio" element 360, but is not duplicated below the entry for the "image" element 358, which also includes a "data" attribute 380. Likewise, entries corresponding to the "x" attribute 386 and "y" attribute 388 of "image" element 358 immediately follow the entry for the "image" element 358. The last two entries (indexes 16 and 17) in the DTD portion of the name table 394 are associated with the "gif" and "ulaw" notations 382 and 384 declared in the DTD 352.

To reconstruct the original SGML document 350 from the tables of the compact tree, the computer requesting the document first reads the last entry (index 13) in the document instance table 450 to identify the corresponding document type definition and the location of the top level element (the "slideshow" element 488) in the document instance table 450. Because the last entry in the document instance table 450 does not refer to an element of the document, no generic identifier or attribute is associated with the entry. Therefore, the corresponding "GI" and "first attribute" column entries are not examined. The client computer obtains the name of the required DTD from the last entry in the CDATA table 392 (index 12). The last entry in the document instance table 450 has at least one corresponding entry in the child table 444, one of which identifies the location of the highest level element in the document instance table 450. In this case, the last entry in the document instance table 450 includes exactly one child element, which is stored as the last entry in the child table 444 (index 12). This child element indicates the location (index 12) of the highest level element ("slideshow") of the compact tree in the document instance table 450.

The computer then reads the document instance table entries for the highest level element to gather information about that element and its children elements. The "GI" field 452 contains a value of "1", indicating that the corresponding generic identifier information is located in the second position (index 1) of the GI declaration table 416. The "identifier" field 418 in the GI declaration table contains a value of "15", indicating that the name of the highest level element ("slideshow") is located in the sixteenth entry (index 15) in the name table 394. The "first attribute" field 420 in the GI declaration table contains a value ("8") greater than the highest index of the attribute declaration table 408, which indicates that the "slideshow" element has no corresponding attributes. The requesting computer then returns to the document instance table and finds that because the "slideshow" element has no corresponding attributes, the "flags" field 454 has no bits set and the "first attribute" field 456 contains an invalid index value ("10"). The "first child" field 458 and the "number of children" field 460 indicate that the "slideshow" element has two sub-elements at the eleventh and twelfth positions in the child table 444. The "value" field 448 of the child table 444 indicates that information for the first of these sub-elements is located in the ninth position (index 8) of the document instance table 450 and that information for the second sub-element is located in the twelfth position (index 11) of the document instance table 450.

Reading the information contained in the ninth entry of the document instance table 450, the computer learns that the generic identifier for the sub-element is contained in the fifth position (index 4) of the GI declaration table 416. This entry in the GI declaration table 416 points the computer to the thirteenth position (index 12) of the name table 394, which indicates that the sub-element is a "slide" element. The attribute fields 420, 422, and 424 of the GI declaration table 416 indicate that each slide element has two associated unfixed attributes that are identified by the seventh and eighth entries (indices 6 and 7) in the "attribute" declaration table 408. The first attribute of each "slide" element is an IDREF-type attribute, the name ("next") of which is contained in the fifteenth entry (index 14) in the name table 394. The "next" attribute has no default value, as indicated by the null string contained in the first entry (index 0) of the CDATA table 392. The second attribute of the "slide" element is an ID-type element, the name of which ("name") is contained in the fourteenth entry (index 13) in the name table 394. The "name" attribute also has a null default value.

After identifying the "name" and "next" attributes associated with the "slide" element, the computer returns to the ninth entry of the document instance table 450 and reads the values of the bits in the corresponding entry of the "flags" field 454. Because each of the first two bits is set, the computer knows it must retrieve the actual values of the "name" and "next" attributes from the attribute value table 440. The "first attribute" field 456 directs the computer to the seventh and eighth positions (indices 6 and 7) of the attribute value table 440, which in turn refers the computer to the twenty-fourth and twenty-fifth positions (indices 23 and 24) of the name table 394. From the name table, the computer learns that the "name" of the first slide element is "first" (index 24) and that the "next" slide element is the "end" slide element (index 23). The "first child" and "number of children" fields 458 and 460 indicate that the "first" slide element has four sub-elements, which are listed sequentially beginning at the fifth position (index 4) of the child table 444. The computer then accesses the value field 448 of the child table 444 to learn that information for the four sub-elements of the "first" slide element is contained in the second, fourth, sixth, and eighth entries (indices 1, 3, 5, and 7) of the document instance table.

The computer next accesses information for the first sub-element of the "first" slide. The corresponding entry in the "GI" field 452 of the document instance table 450 indicates that the generic identifier is identified in the first entry (index 0") in the GI declaration table 416. The "identifier" field 418 of the GI declaration table 416 in turn directs the computer to the seventh entry (index 6) of the name table 394, which identifies the first sub-element as an "item"element. The GI declaration table 416 then directs the computer to the fifth and sixth entries (indices 4 and 5) in the attribute declaration table 408 for information about the two unfixed attributes associated with the "item" element. The first attribute is a CDATA attribute, the name of which ("start") is located in the twelfth position (index 11) of the name table 394. The "start" attribute has a default value of "0" as indicated by the "default" field 414 of the attribute declaration 408 and the second entry (index 1) of the CDATA table 392. The second attribute of the "item" element is the "units" attribute, as indicated by the "name" field 410 of the attribute declaration table 408 and the eighth entry (index 7) in the name table 394. The "units" attribute has three possible values, "seconds", "minutes", and "hours", which are referenced by the enumeration table 404 as the ninth through eleventh positions (indices 8-10) of the name table 394. The default value of the "units" attribute is "seconds".

After identifying the attributes associated with the "item" element, the computer returns to the document instance table 450 and skips the "first attribute" field since none of the bits in the "flags" field are set. The computer then reads information from the "first child" field 458 and the "number of children" field 460 to learn that the first "item" element has one sub-element identified by the first entry (index 0) in the child table. The "value" field 448 of the child table 444 directs the computer to the first entry (index 0) in the document instance table 450 for information about this sub-element. The "GI" field 452 of the document instance table 450 indicates that the generic identifier for the sub-element is identified by the third entry (index 2) of the GI declaration table 416. The "identifier" field 418 of the GI declaration table 416 indicates that this sub-element is an "audio" element, as specified in the second position (index 1) of the name table 394. The "unfixed attribute" field 424 of the GI declaration table 416 indicates that the "audio" element has a single attribute which, according to the "first" field 420 of the GI declaration table 416, is identified by the first entry (index 0) of the attribute declaration table 408. The "name" field 410 of the attribute declaration table 408 and the third entry (index 2) of the name table 394 indicate that this attribute is a "data" attribute. This "data" attribute is an entity having a Null default value, as indicated in the "default" field 414 of the attribute declaration table 408 and the first entry (index 0) of the CDATA table 392. Because the first bit of the corresponding entry in the "flags" field 454 of the document instance table 450 is set, the computer must look to the attribute value table entry (index 0) identified in the "first attribute" field 456 of the document instance table 450 to learn the value of the "data" attribute. This entry in the attribute value table 440 directs the computer to the nineteenth entry (index 18) in the name table 394 (*i.e.*, the first name table entry corresponding to the document instance), which indicates that the name of the "data" attribute is "audio-1". Also, because the attribute is an entity, the computer accesses the first entry (index 0) in the entity table to determine which entity is associated with the "audio-1" attribute. The "type" field 434 of the entity table 430 indicates that the entity is of the type "NDATA", and the "notation" field 436 indicates that the entity associated with the "audio-1" attribute is defined in the second entry (index 2) of the notation table 396. The "name" field of the notation table 396 indicates that the entity has the "ulaw" notation, as listed in the eighteenth entry (index 17) of the name table 394. The notation table 396 also indicates that the "audio-1" attribute is a SYSTEM type object, as indicated in the "value" field 402 of the notation table 396. The "value" field 438 of the entity table 430 then directs the computer to the fifth entry (index 4) of the CDATA table 392 for the name of the entity associated with the "audio-1" attribute. The CDATA table 392 specifies that the "audio-1.ulaw" data file is associated with the "audio-1" attribute of the first "audio" element. The computer returns to the first entry (index 0) of the document instance table 450 and learns that the first "audio" element has no sub-elements.

The computer then moves to the fourth entry (index 3) of the document instance table 450, which represents the second sub-element of the first "slide" element. Like the first sub-element of the "slide" element, the second sub-element is an "item" element having two unfixed attributes, a "start" attribute and a "units" attribute. Because none of the bits in the corresponding entry in the "flags" field 454 of the document instance table 450 is set, both the "start" and the "units" attributes are set to the default values, which means that the second item element begins simultaneously with the first item element. The "first child" field 458 and "number of children" field 460 indicate that the second "item" element also has a single sub-element, which is identified by the second entry (index 1) of the child table 444. Information for the sub-element is contained in the third entry (index 2) of the document instance table 450.

The "GI" field 452 of the document instance table 450, the "identifier" field 418 of the GI declaration table 416, and the fourth entry (index 3) of the name table 394 indicate that the only sub-element of the second "item" element is an "image" element. The "unfixed attribute" field 424 of the GI declaration table 416 indicates that the "image" element has three unfixed attributes, which are listed consecutively beginning with the second entry (index 1) of the attribute declaration table 408. The first attribute of the "image" element is a "data" entity having a null default value. The other two attributes are the "x" and "y" values that indicate the "x-y" position of the "image" element in the presentation space. These attributes are identified in the "name" field 410 of the third and fourth entries (indices 2 and 3) in the attribute declaration table 408 and the fifth and sixth entries (indices 4 and 5) in the name table 394. The "default" field 414 of the attribute declaration table 408 and the second entry (index 1) in the CDATA table indicate that both the "x" and "y" attributes have a default value of "0". The "flags" field 454 indicates that the "x" and "y" attributes of the first "image" element have the default values, while the value of the "data" attribute of the first "image" element is specified by the second entry (index 1) of the attribute value table. The "value" field 442 of the attribute value table 440 and the twentieth entry (index 19) of the name table 394 identify "image-1" as the name of this "data" attribute. The second entry (index 1) of the entity table and the first entry (index 0) in the notation table 396 identify the "image-1" attribute as a SYSTEM type object of the ".gif" file type. The "image-1" attribute is associated with the "image-1.gif" data file, as indicated by the sixth entry (index 5) of the CDATA table 392.

The computer then retrieves information about the other two sub-elements of the first "slide" element from the sixth and eighth entries (indices 5 and 7) in the document instance table 450. The computer learns that each of the sub-elements is a "item" element having a single sub-element. The first of these "item" elements includes an "image" sub-element that is associated with the "PS-Adobe-3.0" data file and that has a presentation delay of two seconds. The second of these "item" elements includes an "audio" sub-element that is associated with the "audio-2.ulaw" data file and that has a presentation delay of 130 seconds.

After the computer displays the second "audio" sub-element of the first "slide" element, it moves to the second "slide" element in the "slideshow" presentation. As discussed above, the second "slide" element is represented by information contained in the twelfth entry (index 11) of the document instance table 450. Because the second "slide" element is the last element of the "slideshow" presentation, the first bit of the "flags" field 454 and the document instance table 450 is cleared and the second bit is set, indicating that the "next" attribute of the "slide" element has the null default value, while the "name" attribute has the value ("end") identified by the last entry (index 9) in the attribute value table 440. The second "slide" element has a single "item" sub-element, which in turn has a single "image" sub-element associated with the "end.gif" data file. The third "image" element is associated with the "end.gif" data file and is displayed at the default "x-y" location (0,0) and with the default presentation delay of zero seconds. Once the second "slide" element has been presented, the requesting computer has fully reconstructed and displayed the "slideshow" document.

The first four entries of the CDATA table 392 represent information in the DTD, so every CDATA field in one of the other tables in FIGURE 11A must contain two bits to reference an entry in the CDATA table 392. Likewise, the first 18 entries of the name table 394 represent information in the DTD, so every name field in the other tables of FIGURE 11A must contain five bits to reference an entry in the name table 394. Therefore, each of the first two entries in the notation table 396, both of which represent information in the DTD, must be eight bits in length: five bits representing the index of the notation name 398 in the name table 394; one bit indicating the notation type 400 (*i.e.*, "SYSTEM" or "PUBLIC"); and two bits representing the index of the notation value 402 in the CDATA table 392. Because the notation table 396 has two entries representing information in the DTD, only one bit is needed in the other tables of FIGURE 11A to access information in the notation table 396. No entities are defined in the DTD of FIGURE 10A, so no entity table is shown in FIGURE 11A.

The DTD of FIGURE 10A defines only a single enumeration with three possible values. Therefore, the enumeration table 404 references only three entries, beginning at the index in the name table 394 indicated by a five-bit NameTable field 406.

Each entry in the attribute declaration table 408 includes three fields: a five-bit name field 410 representing the index of the attribute name in the name table 394; a three-bit type field 412 indicating which of the possible types (name, entity, CDATA, IDREF, ID, or enumeration) each attribute takes on; and a five-bit default value field 414 representing the index of the default value in the name table 394. Because the attribute declaration table 408 has eight entries, every entry in one of the other tables of FIGURE 11A referencing the attribute declaration table 408 must contain four bits.

In the preferred embodiment, each entry in the GI declaration table 416 has 11 bits: five bits representing the generic identifier name 418 in the name table 394; four bits representing the index of the first attribute 420 in the attribute declaration table 408; and two bits representing the number of unfixed attributes 424 (no GI in the DTD has more than three possible unfixed attributes). Because the DTD of FIGURE 10A defines no fixed attributes defined for any GI, no bits are needed in the "number of fixed attributes" field 422.

In addition to the four entries representing the DTD, the CDATA table 392 also includes nine entries representing the document instance. Therefore, any CDATA fields in the document instance tables of FIGURE 11B must contain four bits to reference an entry in the CDATA table 392. The name table 394 contains eight entries representing the document instance, yielding 26 total entries in the name table 394. As a result, five-bit fields are used in the document instance tables of FIGURE 11B to reference the name table 394. The notation table 396 contains one entry representing the document instance, in addition to the two DTD entries, so two-bit fields are used in the document instance tables to reference the notation table 396. In the preferred embodiment, the entry in the notation table 396 representing the document instance uses 10 bits: five bits representing the index of the notation name 398 in the name table 394, one bit indicating the notation type 400 ("PUBLIC"), and four bits representing the index of the notation value 402 in the CDATA table 392.

Because the document instance of FIGURE 10B defines five entities, the compact tree for the document instance includes an entity table 430. Entries in the other document instance tables contain three bits to reference the five entries in the entity table 430. Each entry in the entity table 430 itself contains 13 bits: five representing the index of the entity name 432 in the name table 394; two indicating the entity type 434 (NDATA, SDATA, or CDATA); two representing the index of the entity notation 436 in the notation table 396; and four representing the index of the entity value 438 in the CDATA table 392.

The compact tree for the document instance also includes an attribute value table 440, each entry of which includes a five bit attribute value field 442 representing the index of the attribute name in the name table 394. A child table 444 includes a one-bit field 446 indicating the type of each child entry ("ELEMENT" or "PSEUDO-ELEMENT") and a four-bit field 448 representing the index of the child entry in either the CDATA table 392 or the document instance table 450, discussed below.

The document instance table 450 has 15 bits per entry. Five bits represent the index of the entry's GI 452 in the GI declaration table 416. A three-bit flag vector 454 indicates which, if any, of the three possible unfixed attributes are not set to the corresponding default value. For "slide" elements (GI table index of "4" and name table index of "12"), the first bit in the flag field 454 indicates whether the "name" attribute is set to the default value, and the second bit indicates whether the "next" attribute is set to the default value. The third bit is not used. For "item" elements (GI table index of "0" and name table index of "6"), the first bit indicates whether the "units" attribute is set to the default value, and the second bit indicates whether the "start" attribute is set to the default value. The third bit is not used. For "audio" elements (GI table index of "2" and name table index of "1"), the first bit indicates whether the "data" attribute is set to the default value, and the other two bits are not used. For " image" elements (GI table index of "3" and name table index of "3"), the first bit indicates whether the "data" attribute is set to the default value, the second and third bits indicate whether the "x" and "y" attributes, respectively, are set to the corresponding default values.

Each entry in the document instance table 450 also contains four bits representing the index for the attribute value table 440 entry holding the actual value of the first unfixed attribute 456 that is not set to the default value. Four bits represent the index of the entry's first child 458 in the child table 444. Because no element in the document instance has more than four children, three bits are used to indicate the number of children 460 for each entry in the instance table 450.

In this example, the actual DTD file uses approximately 455 bytes of storage space, while the compact tree representation of the DTD consumes only 128 bytes. Likewise, the actual document instance file requires approximately 731 bytes of storage space, while the compact tree representation of the document instance uses only 183 bytes.

Referring to FIGURE 12, a sample SGML document instance 550 generated according to the DTD 352 of FIGURE 10A includes a top-level "slideshow" element 552 and three "slide" sub-elements: a "first" slide element 554, a "second" slide element 556, and an "end" slide element 558. The simple structure of the document instance 550 lends the corresponding parse tree naturally to partitioning at each slide element, though the parse tree also could be partitioned in other ways.

Referring to FIGURES 13A and 13B, the compact tree representations of the DTD 352 and the document instance 550 include a name table 560, a notation table 562, a CDATA table 564, an enumeration table 566, an attribute declaration table 568, a GI declaration table 570, a document instance table 572, an entity table 574, an attribute value table 576, and a child table 578, all similar to those described above. The CT representation for the document instance also includes an "element boundaries" table 580, which identifies the entries in the attribute value, child, and document instance tables that correspond to each of four partitions, or sub-trees, in the CT representation of the document instance. The first partition 582 ("top") identified in the element boundaries table 580 includes only the top-level elements of the document: the "slideshow" element 552 and each of the "slide" sub-elements 554, 556, and 558. The next partition 584 ("first") in the element boundaries table 580 includes only those sub-elements associated with the "first" slide element 554. Likewise, the next partition 586 ("second") includes only those sub-elements associated with the "second" slide element 556, and the last partition 588 ("end") includes only those sub-elements associated with the "end" slide element 558.

For each sub-tree, the element boundaries table 580 includes seven information fields: an "element" field 590 identifying the sub-tree; an "attribute value start" field 592 and an "attribute value end" field 594 identifying the indices of the first and last entries, respectively, in the attribute value table 576 corresponding to the sub-tree; a "child start" field 596 and a "child end" field 598 identifying the indices of the first and last entries, respectively, in the child table 578 corresponding to the sub-tree; and a "document instance start" field 600 and a document instance end" field 602 identifying the indices of the first and last entries, respectively, in the document instance table 572 corresponding to the sub-tree.

When the requesting computer first requests partitioned delivery of the document, the CDATA table 564, the name table 560, the notation table 562, and the entity table 574 are sent in their entirety, but only the portions corresponding to the "top" sub-tree 582 are sent for the attribute value table 576 (indices 10 through 14), the child table 578 (indices 8 through 19), and the document instance table 572 (indices 16 through 20). When the requesting computer later requests a child element of one of the "slide" elements, only those entries in the attribute value, child, and document instance tables corresponding to that "slide" element are sent. For example, if a child element of the "second" slide element 556 is requested, the requesting computer receives only the seventh through ninth entries (indices 6 through 8) from the attribute value table 576, the fifth through seventh entries (indices 4 through 6) from the child table 578, and the ninth through fourteenth entries (indices 8 through 13) from the document instance table 572.

Implementation

The method of the invention may be implemented in hardware or software, or a combination of both. However, preferably, the method of the invention is implemented in computer programs executing on programmable processors each comprising a processor, a data storage system (including volatile and non-volatile memory and/or storage elements), at least one input device, and at least one output device.

Each program is preferably implemented in a high level procedural or object oriented programming language to communicate with a processor. However, the programs can be implemented in assembly or machine language, if desired. In any case, the language may be a compiled or interpreted language.

Each such computer program is preferably stored on a storage media or device (*e.g.*, ROM, flash RAM, or magnetic diskette) readable by a general or special purpose programmable processor, for configuring and operating the processor when the storage media or device is read by the processor to perform the procedures described herein. The inventive system may also be considered to be implemented as a processor-readable storage medium, configured with a computer program, where the storage medium so configured causes a processor to operate in a specific and predefined manner to perform the functions described herein.

A number of embodiments of the present invention have been described. Nevertheless, it will be understood that various modifications may be made without departing from the spirit and scope of the invention. For example, the CT instance object may have additional headers that provide information such as character set identification, sequencing of objects, time stamps, ownership, and other parameters needed for managing or optimizing a distributed hypermedia system. Also, the CT DTD object and the CT instance object may be stored and transmitted with different formats than those shown here. Furthermore, structured document syntaxes other than SGML may be supported, either by handling non-SGML documents directly or by first translating non-SGML documents into the SGML syntax. Additionally, while an implementation has been described that uses a "client request" model, other models may be used, such as a subscriber or broadcast model (client requests a subscription to information, and the server from time to time publishes that information to the client without further requests), or a server push model (a client registers for specific information, and the server from time to time sends such specific information without further request), or combinations of such models.

Accordingly, it is to be understood that the invention is not to be limited by the specific illustrated embodiments, but only by the scope of the following claims.

CLAIMS

What is claimed is:

1. A computer network comprising:
   (a) a client computer that authorizes receipt of a structured document, and is capable of receiving such structured document; and
   (b) a server computer that, upon receiving at least an initial receipt authorization from a client computer, retrieves the structured document, parses the document into a structural portion and a content portion, and thereafter sends at least some of the two portions to the client computer.

2. The computer network of claim 1 wherein the server also sends the client computer a document type definition corresponding to the requested document.

3. The computer network of claim 1 wherein the server compresses the structured portion of the document into a compact tree.

4. The computer network of claim 1 wherein the server compresses the content portion of the document.

5. The computer network of claim 1 further comprising a processor in the client that traverses the structured portion and the content portion of the document to reconstruct the document.

6. The computer network of claim 1 wherein the structured portion of the document is arranged to allow incremental access by the client computer.

7. The computer network of claim 1 wherein the structured portion of the document comprises a compact tree, at least one element of which comprises an associated sub-tree representing sub-elements hierarchically subordinate to the element.

8. The apparatus of claim 7 wherein the compact tree comprises a table defining the structural elements of the document.

9. The computer network of claim 1 wherein the structured document adheres to a formalized syntax.

10. The computer network of claim 9 wherein the syntax comprises SGML.

11. A computer data structure for storing structured document data in a persistent object storage of a network server computer, comprising a compact tree including a parse tree representing the structural hierarchy of the structured document, and a content list representing the data contained in the structured document.

12. A persistent object storage in a network server computer configured to store structured document data as a compact tree comprising a parse tree representing the structural hierarchy of the structured document, and a content list representing the data contained in the structured document.

13. A method of storing information representing a structured document in a persistent object storage of a network server computer, the method comprising the steps of :

(a) parsing the structured document to form a parse tree representing the structural hierarchy of the structured document and a content list representing data contained in the structured document, and

(b) storing the parse tree and the content list in the persistent object storage.

14. The method of claim 13 further comprising the step of storing in the persistent object storage a document type definition corresponding to the structured document.

15. The method of claim 13 further comprising the step of compressing the content list before storing it in the persistent object storage.

16. A method of sharing a structured document in a computer network, the method comprising the steps of :

(a) transmitting a parse tree representing the structural hierarchy of the structured document and a content list representing data contained in the structured document from a network server computer to a network client computer,

(b) reconstructing the structured document in the network client computer using the transmitted parse tree and content list.

17. The method of claim 16 further comprising the step of transmitting a document type definition corresponding to the structured document from the network server computer to the network client computer.

18. The method of claim 16 wherein the content list comprises compressed text data.

19. The method of claim 18 further comprising the step of expanding the compressed text data after transmitting it to the network client computer.

20. The method of claim 16 wherein the parse tree and the content list are transmitted incrementally to the network client computer.

21. The method of claim 16 wherein the parse tree is transmitted to the network client computer as partitioned sub-trees.

22. A method of displaying a structured document in a network client computer, the method comprising the steps of :

(a) receiving from a network server computer a parse tree representing the structural hierarchy of the structured document and a content list representing data contained in the structured document,

(b) reconstructing the structured document in the network client computer using the received parse tree and content list, and

(c) displaying the reconstructed structured document on a display device attached to the network client computer.

23. A computer program, residing on a computer-readable medium, comprising instructions for causing a server processor to parse a structured document data into a compact tree comprising a parse tree representing the structural hierarchy of the structured document, and a content list representing the data contained in the structured document.

24. A computer program, residing on a computer-readable medium, for representing a structured document in a persistent object storage of a network server computer, comprising instructions for causing a processor to:

(a) parse the structured document to form a parse tree representing the structural hierarchy of the structured document and a content list representing data contained in the structured document, and

(b) store the parse tree and the content list in the persistent object storage.

25. A computer program, residing on a computer-readable medium, for sharing a structured document in a computer network, comprising instructions for causing at least one processor to:

(a) transmit a parse tree representing the structural hierarchy of the structured document and a content list representing data contained in the structured document from a network server computer to a network client computer,

(b) reconstruct the structured document in the network client computer using the transmitted parse tree and content list.

26. A computer program, residing on a computer-readable medium, for displaying a structured document in a network client computer, comprising instructions for causing a processor to:

(a) receive from a network server computer a parse tree representing the structural hierarchy of the structured document and a content list representing data contained in the structured document,

(b) reconstruct the structured document in the network client computer using the received parse tree and content list, and

(c) display the reconstructed structured document on a display device attached to the network client computer.

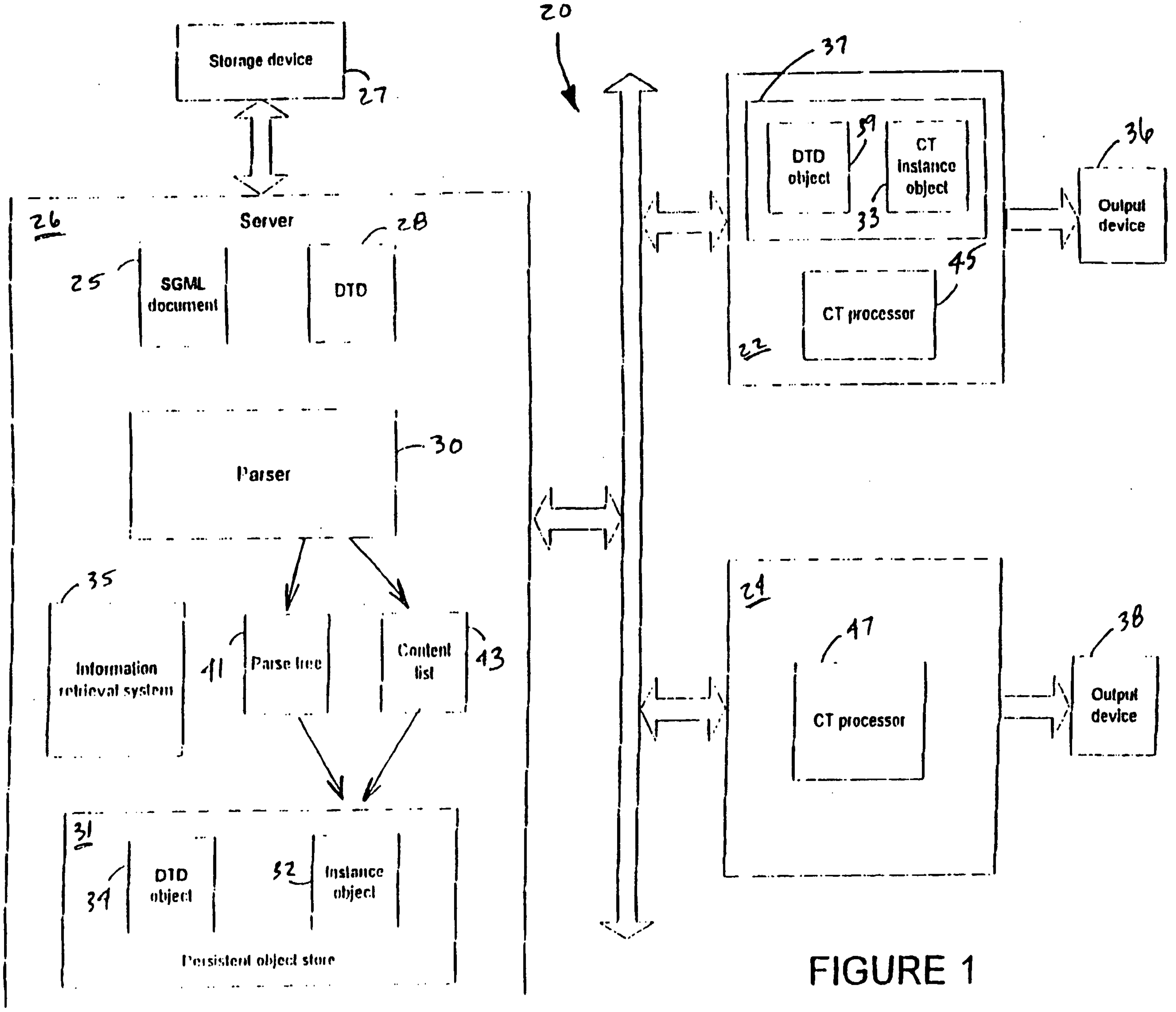

20
Storage device
27
26
Server
25
SGML document
28
DTD
Parser
30
35
Information retrieval system
41
Parse tree
Content list
43
31
34
DTD object
32
Instance object
Persistent object store
37
DTD object
39
CT instance object
33
45
CT processor
22
36
Output device
24
47
CT processor
38
Output device

FIGURE 1

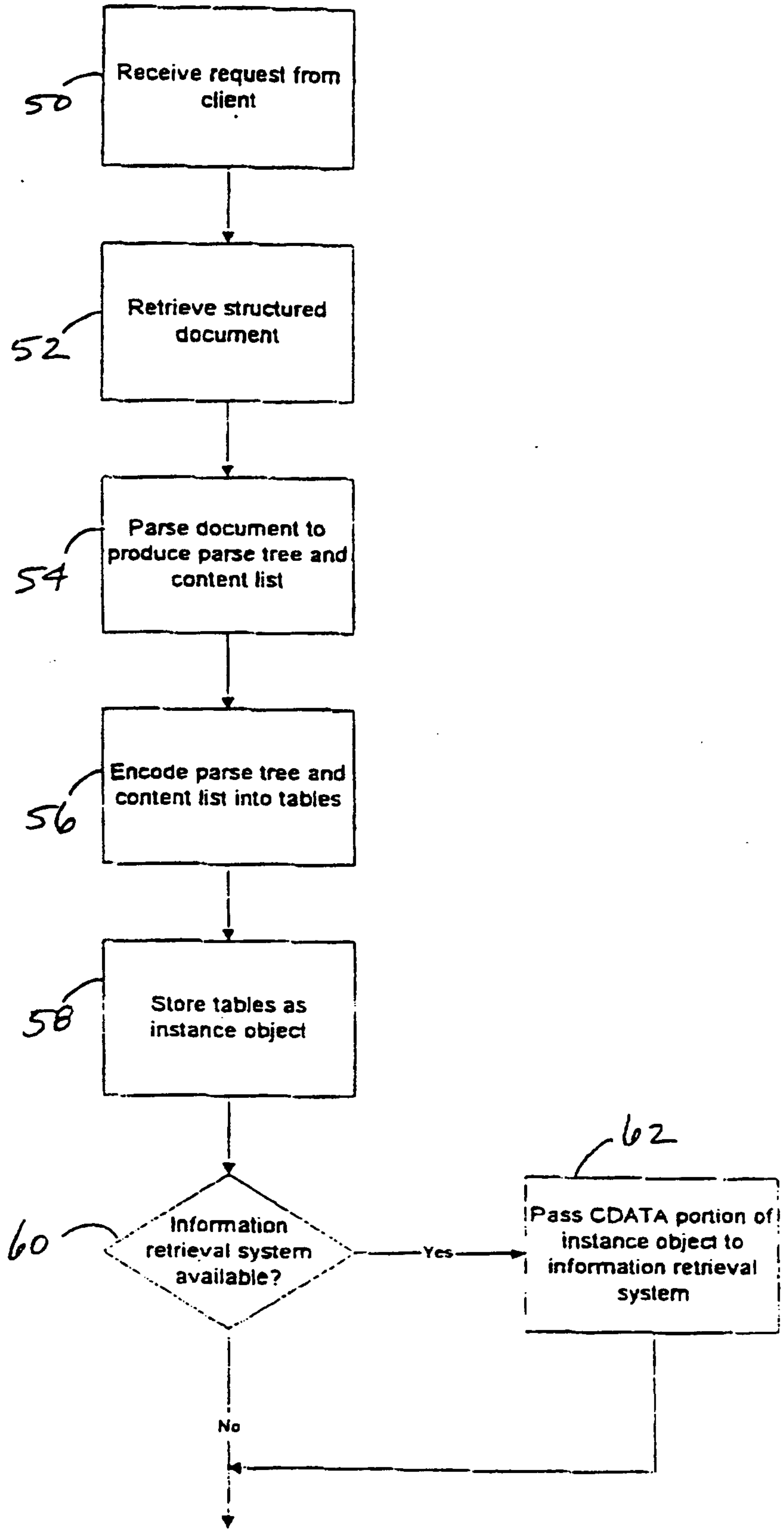

Receive request from client
50
Retrieve structured document
52
Parse document to produce parse tree and content list
54
Encode parse tree and content list into tables
56
Store tables as instance object
58
Information retrieval system available?
60
Yes
Pass CDATA portion of instance object to information retrieval system
62
No

FIGURE 2A

TO FIGURE 2B

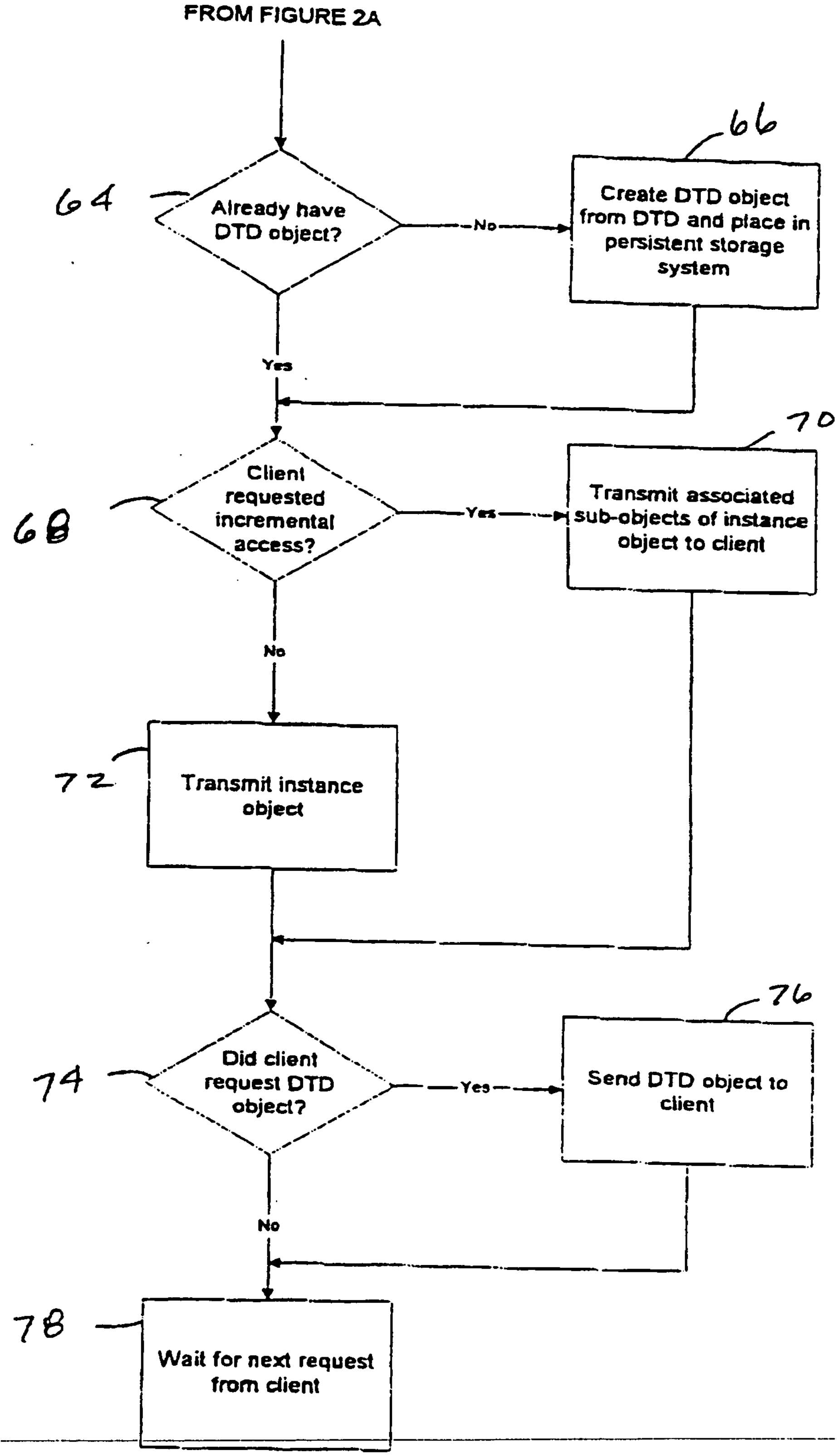

FROM FIGURE 2A
64
Already have DTD object?
No
66
Create DTD object from DTD and place in persistent storage system
Yes
68
Client requested incremental access?
Yes
70
Transmit associated sub-objects of instance object to client
No
72
Transmit instance object
74
Did client request DTD object?
Yes
76
Send DTD object to client
No
78
Wait for next request from client

FIGURE 2B

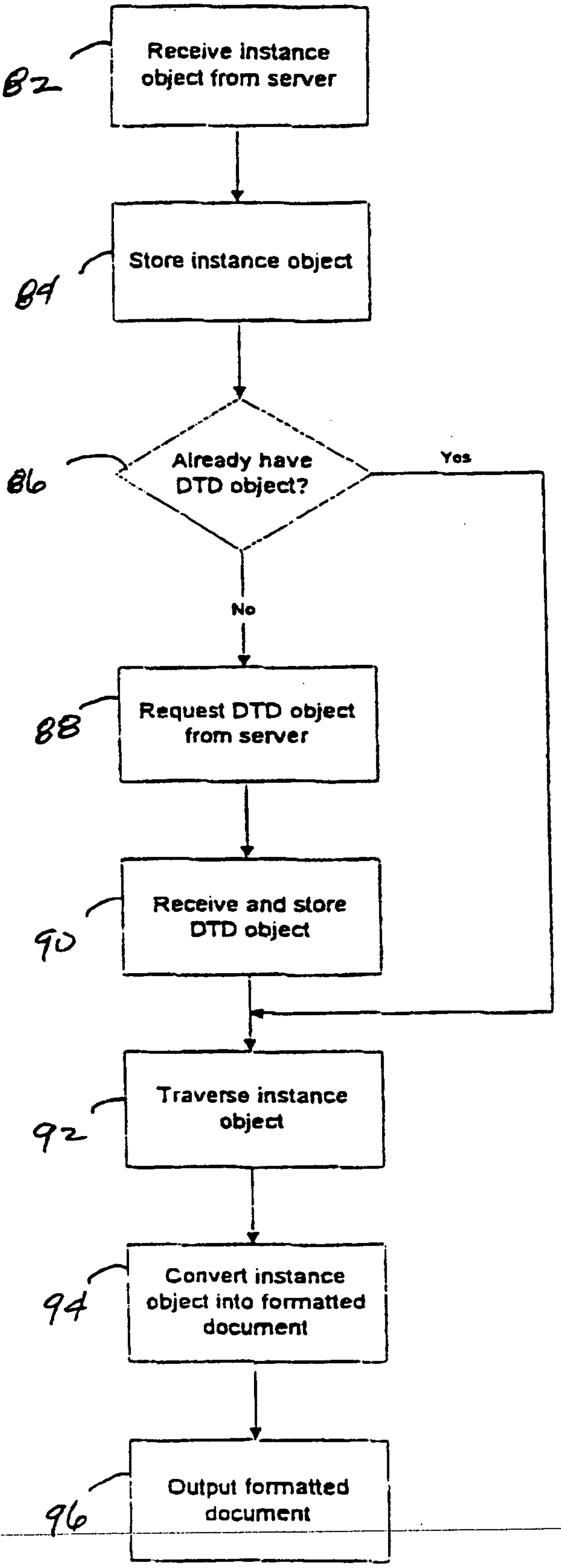
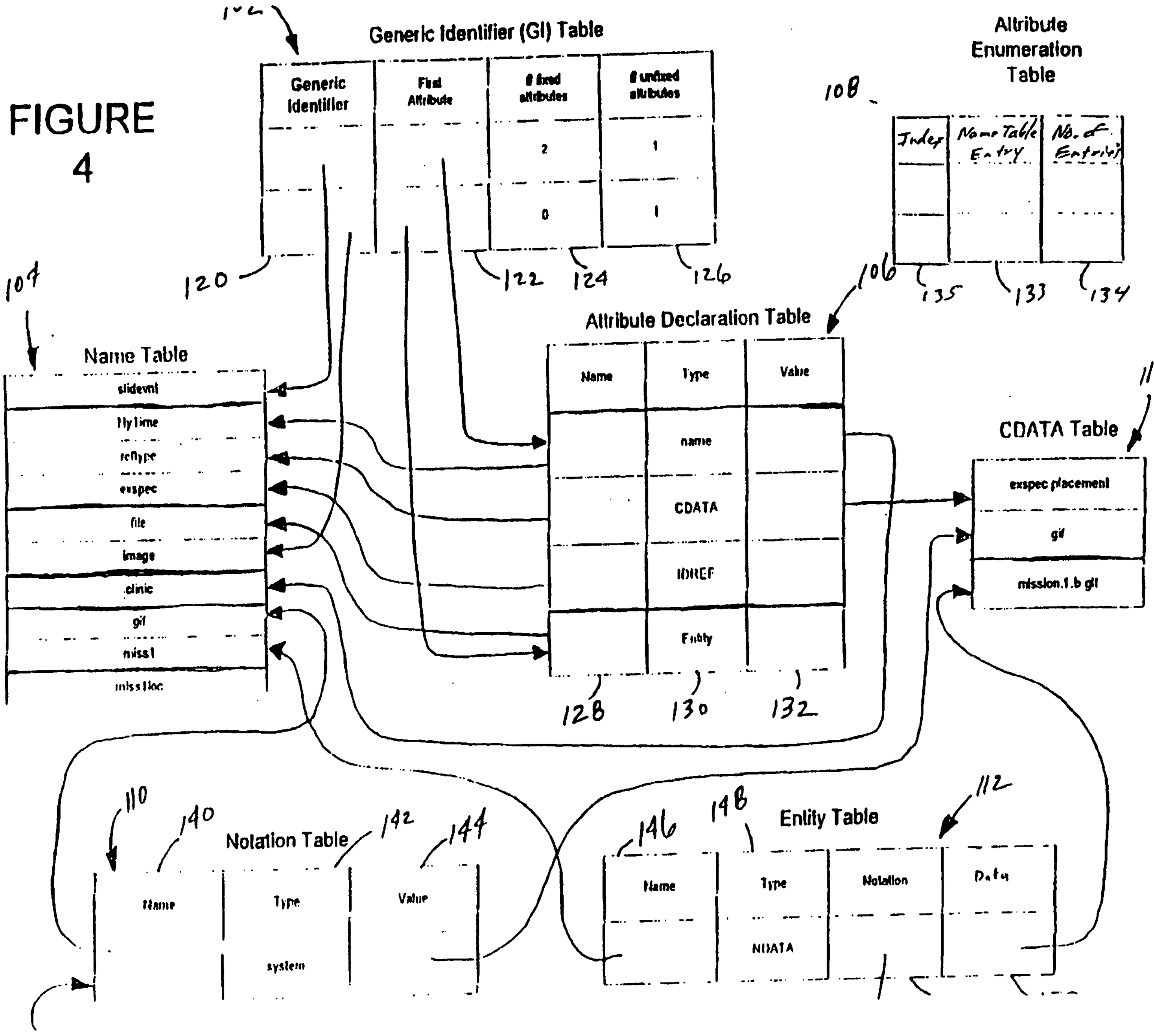

82
Receive instance object from server
84
Store instance object
86
Already have DTD object?
Yes
No
88
Request DTD object from server
90
Receive and store DTD object
92
Traverse instance object
94
Convert instance object into formatted document
96
Output formatted document

FIGURE 3

FIGURE 4


102
Generic Identifier (GI) Table
Generic Identifier
First Attribute
# fixed attributes
# unfixed attributes
2
1
0
1
120
122
124
126
108
Attribute Enumeration Table
Index
Name Table Entry
No. of Entries
135
133
134
104
Name Table
slidevnt
flytime
reftype
exspec
file
image
clinic
gif
miss1
miss1loc
106
Attribute Declaration Table
Name
Type
Value
name
CDATA
IDREF
Entity
128
130
132
114
CDATA Table
exspec placement
gif
mission.1.b gif
110
140
Notation Table
142
144
Name
Type
Value
system
146
148
Entity Table
112
Name
Type
Notation
Data
NDATA

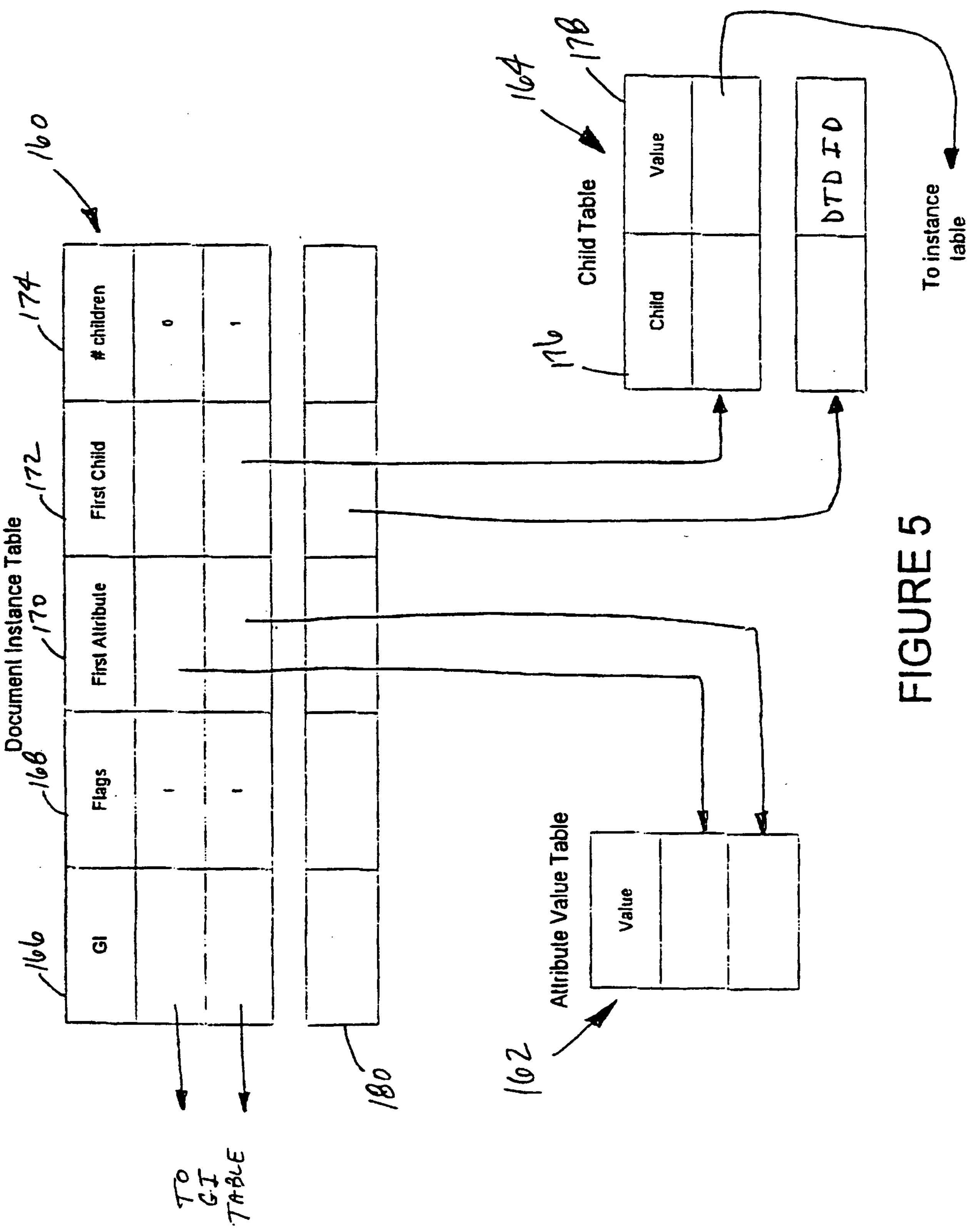

160
Document Instance Table
166
GI
168
Flags
170
First Attribute
172
First Child
174
# children
1
1
0
1
180
TO
G.I
TABLE
162
Attribute Value Table
Value
164
178
Child Table
Child
Value
176
DTD ID
To instance table

FIGURE 5

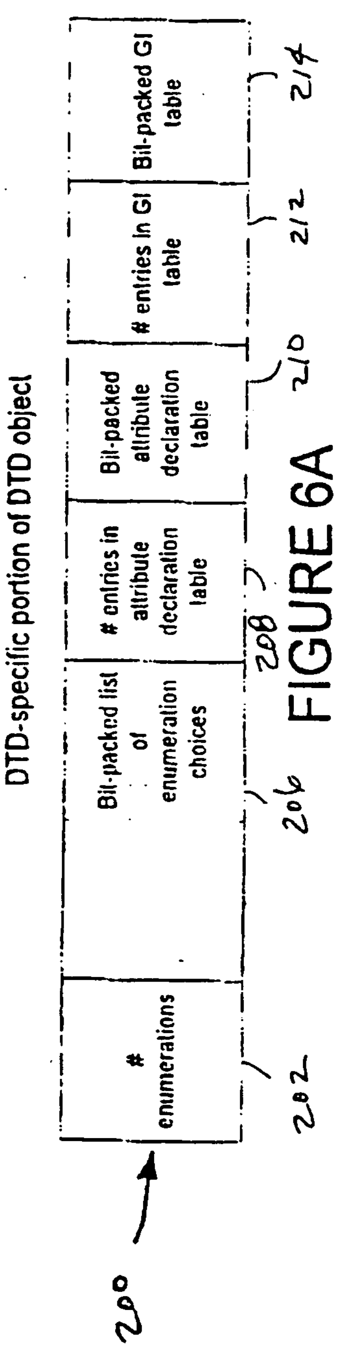

DTD-specific portion of DTD object
# enumerations
Bit-packed list of enumeration choices
# entries in attribute declaration table
Bit-packed attribute declaration table
# entries in GI table
Bit-packed GI table
200
202
206
208
210
212
214

FIGURE 6A

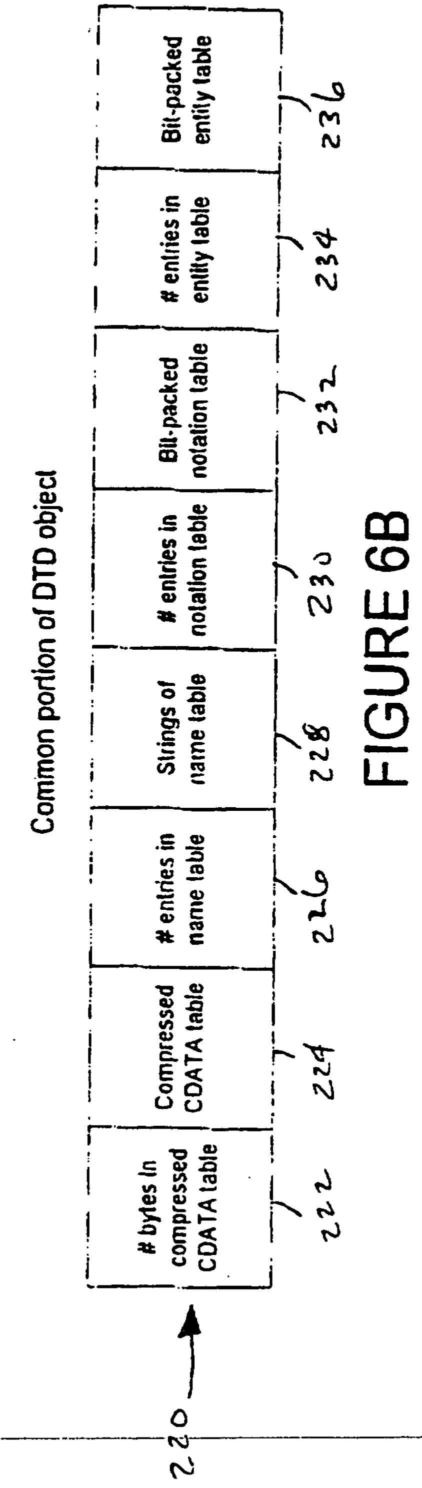

Common portion of DTD object
# bytes in compressed CDATA table
Compressed CDATA table
# entries in name table
Strings of name table
# entries in notation table
Bit-packed notation table
# entries in entity table
Bit-packed entity table
220
222
224
226
228
230
232
234
236

FIGURE 6B

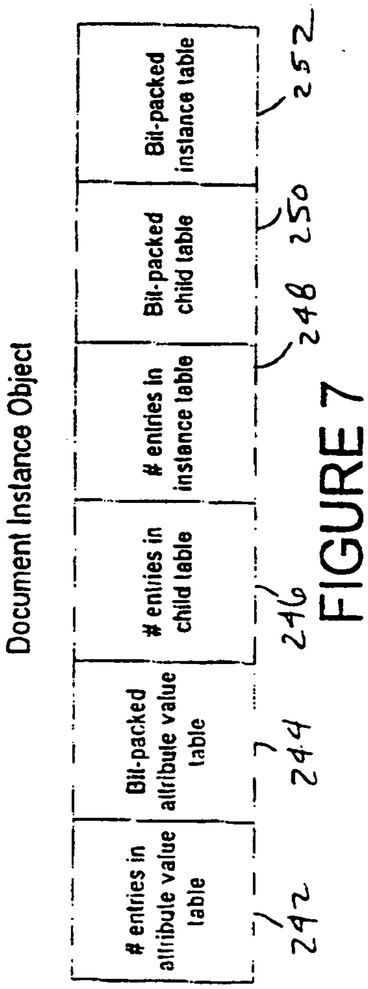

Document Instance Object
# entries in attribute value table
Bit-packed attribute value table
# entries in child table
# entries in instance table
Bit-packed child table
Bit-packed instance table
242
244
246
248
250
252

FIGURE 7

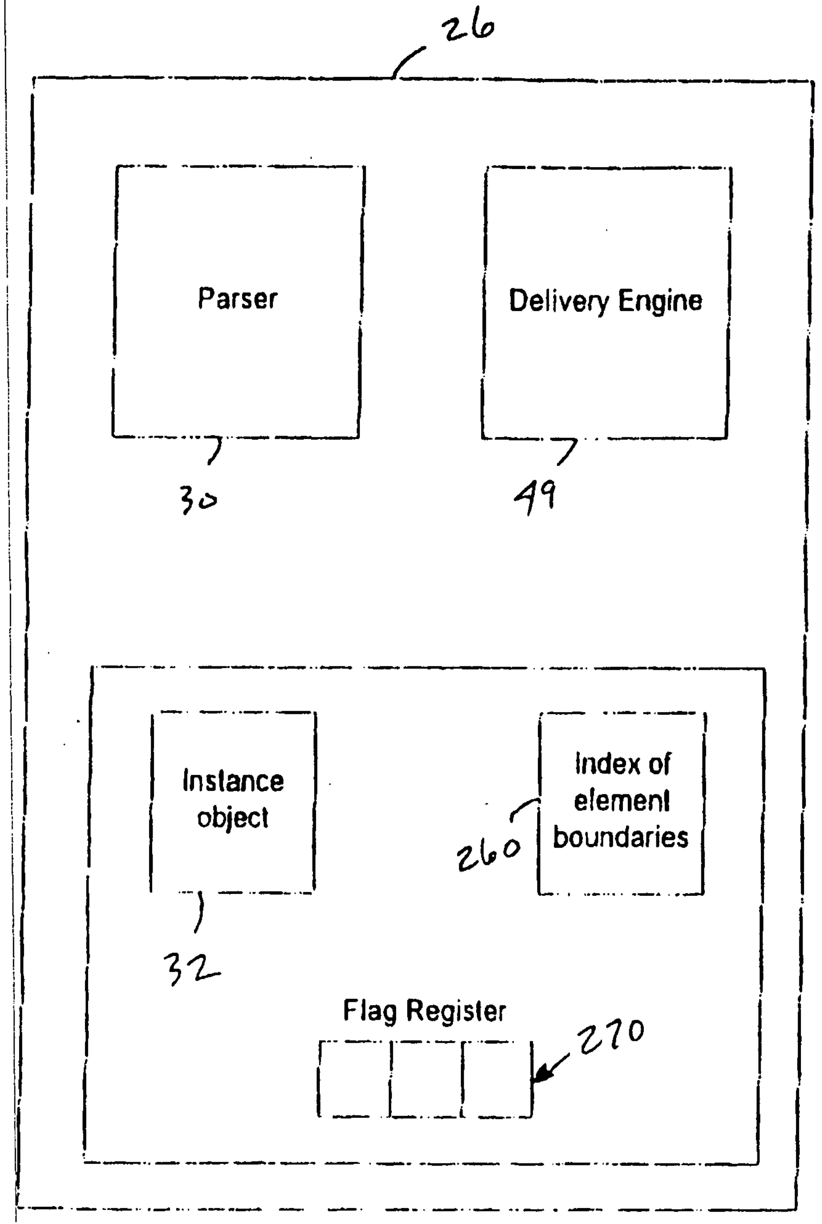


FIGURE 8A

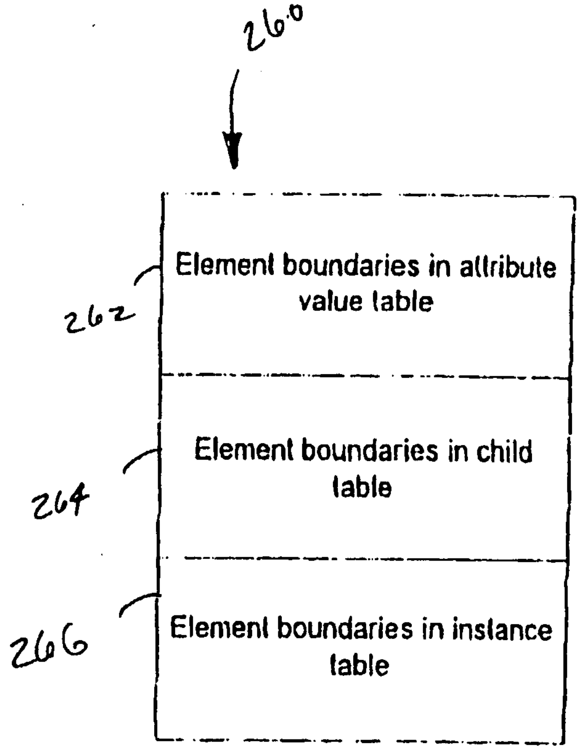

260
Element boundaries in attribute value table
262
Element boundaries in child table
264
Element boundaries in instance table
266

FIGURE 8B

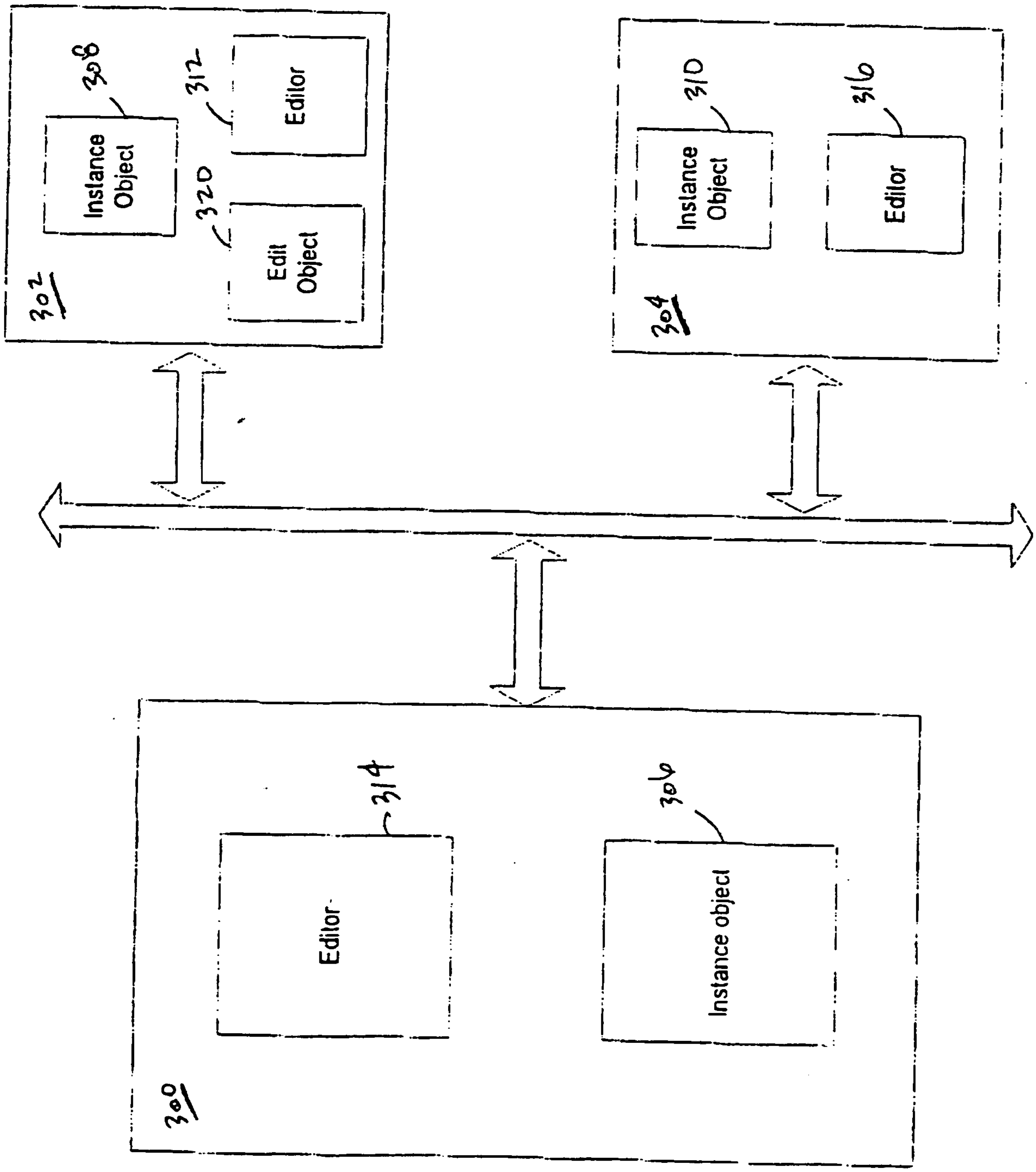

302
308
Instance Object
312
Editor
320
Edit Object
304
310
Instance Object
316
Editor
300
314
Editor
306
Instance object

FIGURE 9A

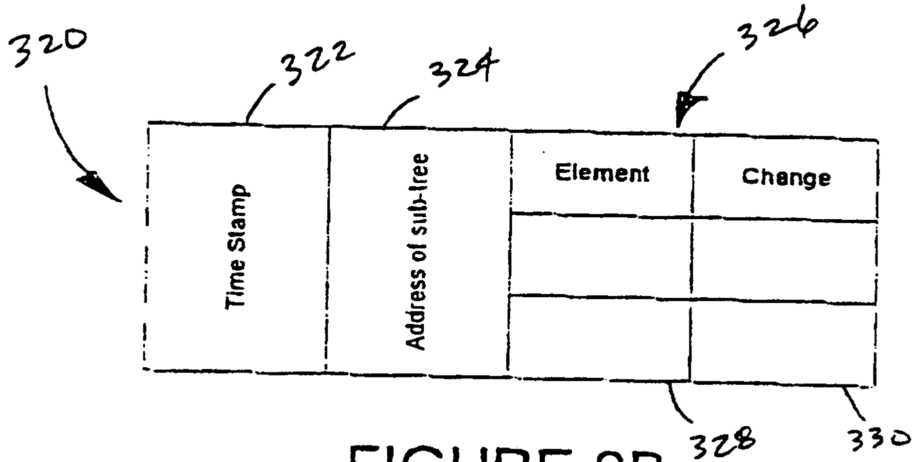
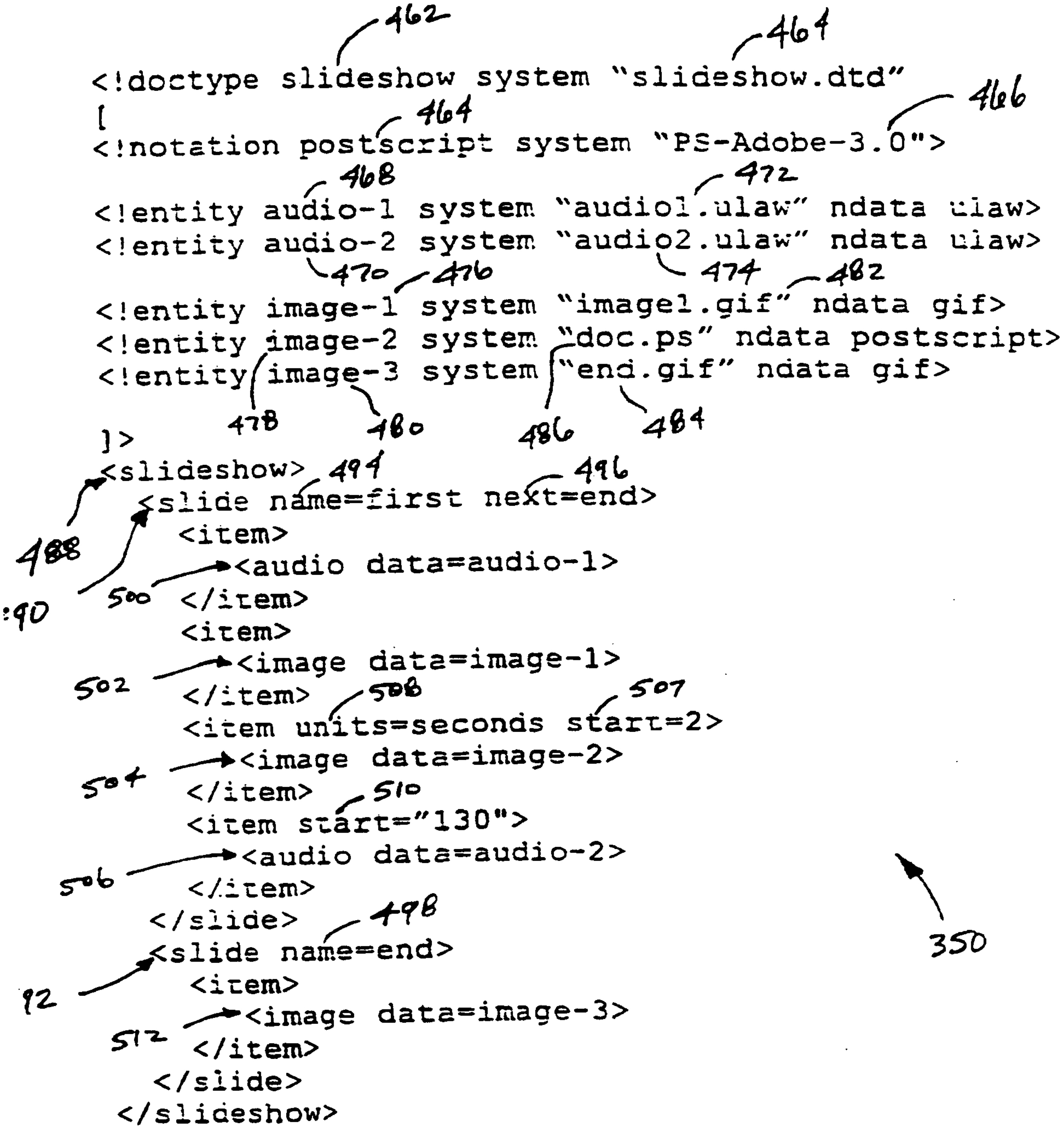

320
322
324
326
Time Stamp
Address of sub-tree
Element
Change
328
330

FIGURE 9B

```
                          353 354
         <!element slideshow - - (slide+)>
         <!element slide - - (item+)>
         <!attlist slide
362 ——name id #required
364 ——next idref #implied
         >              356
         <!element item - - (audio | image)>
         <!attlist item   372      374     376       378
370 ——units (seconds, minutes, hours) seconds
366 ——start CDATA "0" 368
         >
         <!element audio - o empty> 360
         <!attlist audio
381 —— data entity #required
         >
         <!element image - o empty> 358
         <!attlist image
380 ——data entity #required
386 ——x CDATA "0" 390
388 ——y CDATA "0" 391
         >
382 ——<!notation gif system "gif">
        <!notation ulaw system "ulaw">
384

352
```

FIGURE 10A



```
<!doctype slideshow system “slideshow.dtd”
[
<!notation postscript system “PS-Adobe-3.0">
<!entity audio-1 system “audio1.ulaw” ndata ulaw>
<!entity audio-2 system “audio2.ulaw” ndata ulaw>
<!entity image-1 system “image1.gif” ndata gif>
<!entity image-2 system “doc.ps” ndata postscript>
<!entity image-3 system “end.gif” ndata gif>

]>
<slideshow>
 <slide name=first next=end>
   <item>
     <audio data=audio-1>
   </item>
   <item>
     <image data=image-1>
   </item>
   <item units=seconds start=2>
     <image data=image-2>
   </item>
   <item start=”130”>
     <audio data=audio-2>
   </item>
  </slide>
  <slide name=end>
   <item>
     <image data=image-3>
   </item>
  </slide>
 </slideshow>
```

462
464
464
466
468
472
470
476
474
482
478
480
486
484
494
496
488
490
500
502
508
507
504
510
506
498
492
512
350

FIGURE 10B

Name Table 394

Index	Value
0	
1	audio
2	data
3	image
4	x
5	y
6	item
7	units
8	seconds
9	minutes
10	hours
11	start
12	slide
13	name
14	next
15	slideshow
16	gif
17	ulaw
18	audio-1
19	image-1
20	postscript
21	image-2
22	audio-2
23	end
24	first
25	image-3

392

CDATA Table

Index	Value
0	" "
1	"0"
2	"gif"
3	"ulaw"
4	"audio1.ulaw"
5	"image1.gif"
6	"PS-Adobe-3.0"
7	"doc.ps"
8	"2"
9	"audio2.ulaw"
10	"130"
11	"end.gif"
12	"slideshow.dtd"

404 Enumeration Table

Index	Name Table Entry	No. of Entries
0	8	3

406

Notation Table 396

Index	Name	Type	Value
0	16	SYSTEM	2
1	17	SYSTEM	3
2	20	PUBLIC	6

398 400 402

Attribute Declaration Table 408

Index	Name	Type	Default
0	2	ENTITY	0
1	2	ENTITY	0
2	4	CDATA	1
3	5	CDATA	1
4	11	CDATA	1
5	7	ENUM(0)	0
6	14	IDREF	0
7	13	ID	0

410 412 414

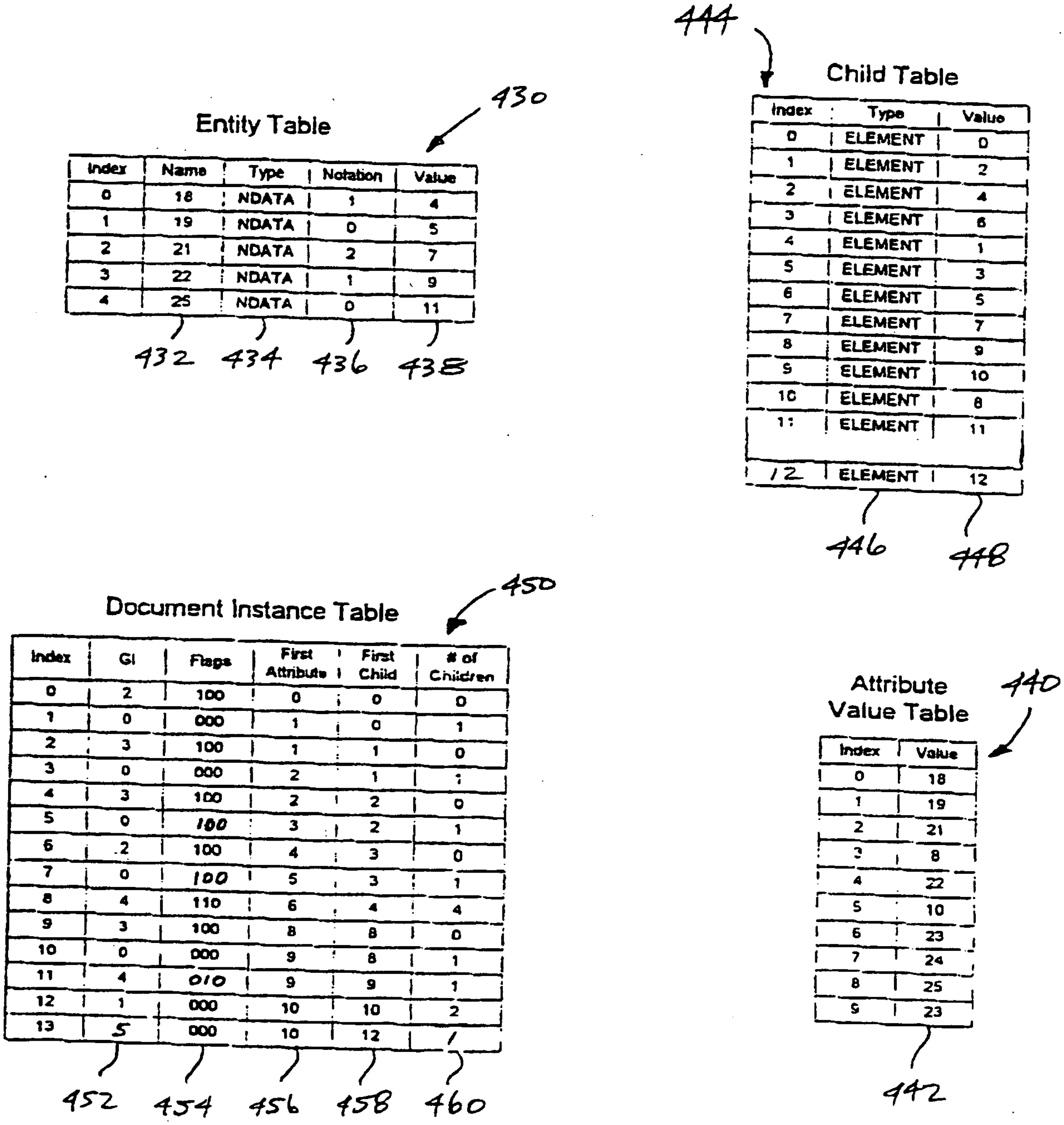
416

GI Declaration Table

Index	Identifier	first	# fixed	# unfixed
0	6	4	0	2
1	15	8	0	0
2	1	0	0	1
3	3	1	0	3
4	12	6	0	2

418 420 422 424

FIGURE 11A


430
432
434
436
438
444
446
448
450
452
454
456
458
460
440
442

Entity Table

Index	Name	Type	Notation	Value
0	18	NDATA	1	4
1	19	NDATA	0	5
2	21	NDATA	2	7
3	22	NDATA	1	9
4	25	NDATA	0	11

Child Table

Index	Type	Value
0	ELEMENT	0
1	ELEMENT	2
2	ELEMENT	4
3	ELEMENT	6
4	ELEMENT	1
5	ELEMENT	3
6	ELEMENT	5
7	ELEMENT	7
8	ELEMENT	9
9	ELEMENT	10
10	ELEMENT	8
11	ELEMENT	11
12	ELEMENT	12

Document Instance Table

Index	GI	Flags	First Attribute	First Child	# of Children
0	2	100	0	0	0
1	0	000	1	0	1
2	3	100	1	1	0
3	0	000	2	1	1
4	3	100	2	2	0
5	0	100	3	2	1
6	2	100	4	3	0
7	0	100	5	3	1
8	4	110	6	4	4
9	3	100	8	8	0
10	0	000	9	8	1
11	4	010	9	9	1
12	1	000	10	10	2
13	5	000	10	12	1

Attribute Value Table

Index	Value
0	18
1	19
2	21
3	8
4	22
5	10
6	23
7	24
8	25
9	23

FIGURE 11B

Name Table

560

Index	Value
0	
1	audio
2	data
3	image
4	x
5	y
6	item
7	units
8	seconds
9	minutes
10	hours
11	start
12	slide
13	name
14	next
15	slideshow
16	gif
17	ulaw
18	audio-1-1
19	image-1-1
20	postscript
21	image-1-2
22	audio-1-2
23	second
24	first
25	audio-2-1
26	image-2-1
27	image-2-2
28	end
29	image-3-1

CDATA Table

564

Index	Value
0	""
1	"0"
2	"gif"
3	"ulaw"
4	"audio1.ulaw"
5	"image1.gif"
6	"PS-Adobe-3.0"
7	"doc.ps"
8	"2"
9	"audio2.ulaw"
10	"130"
11	audio3.ulaw
12	image2.gif
13	image3.gif
14	45
15	end.gif
16	slideshow.dtd

Enumeration Table

566

Index	Name Table	No. of Entries
0	8	3

Notation Table

562

Index	Name	Type	Value
0	16	SYSTEM	2
1	17	SYSTEM	3
2	20	PUBLIC	6

Attribute Declaration Table

568

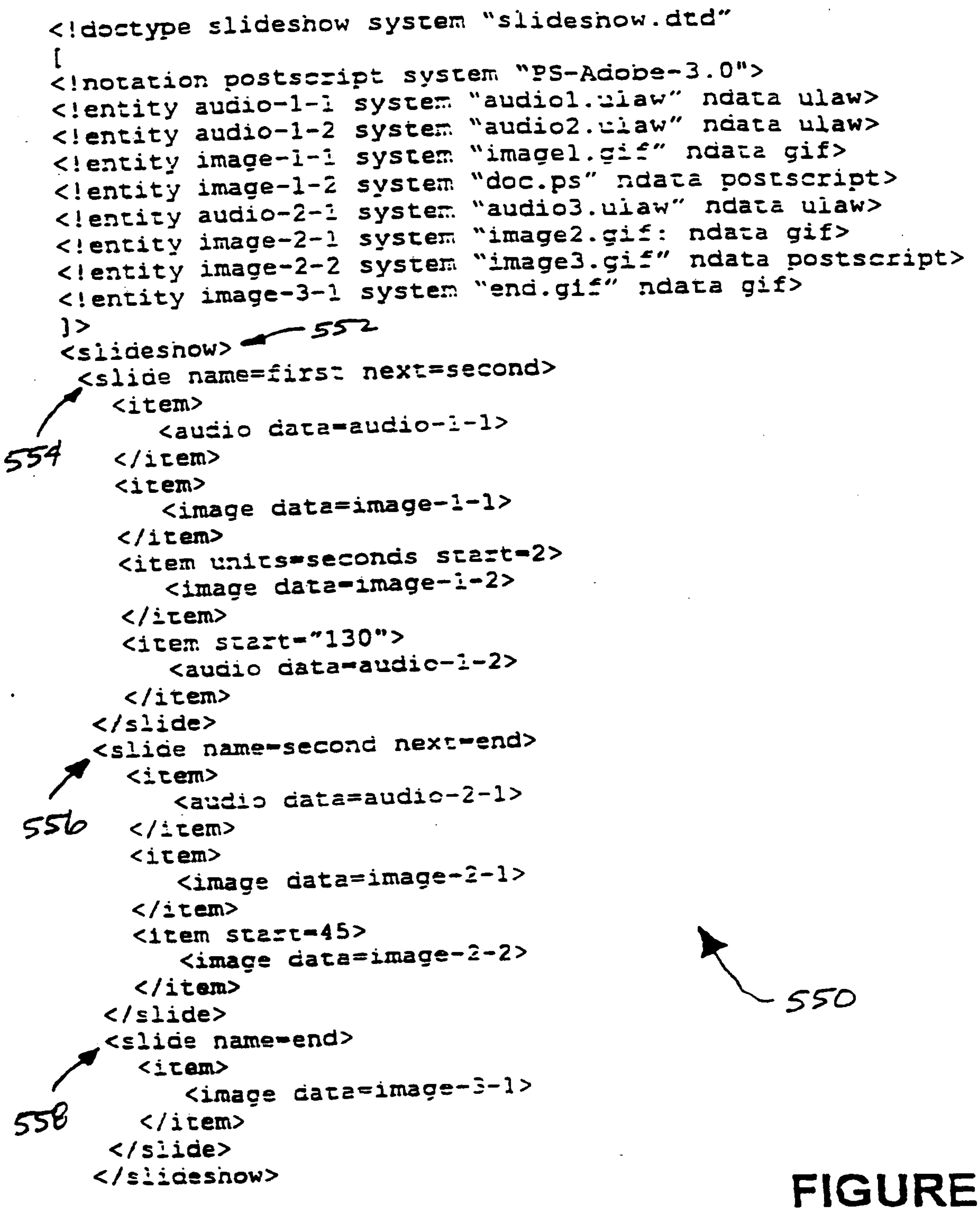
Index	Name	Type	Default
0	2	ENTITY	0
1	2	ENTITY	0
2	4	CDATA	1
3	5	CDATA	1
4	11	CDATA	1
5	7	ENUM(0)	0
6	14	IDREF	0
7	13	ID	0

570

GI Declaration Table

Index	Identifier	first	# fixed	# unfixed
0	6	4	0	2
1	15	8	0	0
2	1	0	0	1
3	3	1	0	3
4	12	6	0	2

FIGURE 13A



```
<!doctype slideshow system "slideshow.dtd"
[
<!notation postscript system "PS-Adobe-3.0">
<!entity audio-1-1 system "audio1.ulaw" ndata ulaw>
<!entity audio-1-2 system "audio2.ulaw" ndata ulaw>
<!entity image-1-1 system "image1.gif" ndata gif>
<!entity image-1-2 system "doc.ps" ndata postscript>
<!entity audio-2-1 system "audio3.ulaw" ndata ulaw>
<!entity image-2-1 system "image2.gif: ndata gif>
<!entity image-2-2 system "image3.gif" ndata postscript>
<!entity image-3-1 system "end.gif" ndata gif>
]>
<slideshow>
 <slide name=first next=second>
   <item>
      <audio data=audio-1-1>
   </item>
   <item>
      <image data=image-1-1>
   </item>
   <item units=seconds start=2>
      <image data=image-1-2>
   </item>
   <item start="130">
      <audio data=audio-1-2>
   </item>
 </slide>
 <slide name=second next=end>
   <item>
      <audio data=audio-2-1>
   </item>
   <item>
      <image data=image-2-1>
   </item>
   <item start=45>
      <image data=image-2-2>
   </item>
 </slide>
 <slide name=end>
   <item>
      <image data=image-3-1>
   </item>
 </slide>
</slideshow>
```

FIGURE 12

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US97/04574

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :G06F 15/00, 17/30
US CL :395/774, 615
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 395/774, 615, 806, 807

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

NONE

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

Please See Extra Sheet.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	GOLDFARB, C.F. HyTime: A standard for structured hypermedia interchange. COMPUTER. August 1991, Vol. 24, No. 8, pages 81-84, especially p. 84, column 2, lines 5-22.	1-26
A	NEWCOMB, S.R. Multimedia interchange using SGML/HyTime. Part I: Structures. IEEE Multimedia. Summer 1995, Vol. 2, No. 2, pages 86-89.	1-26
A	LUBICH, H.P. A Proposed Extension of the ODA Document Model for the Processing of Multimedia Documents. Proceedings of TRICOMM '91: IEEE Conf. on Comm. Software: Comm. for Distributed Applications and Systems. 18 April 1991, pages 59-72.	1-26

□ Further documents are listed in the continuation of Box C. □ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)		
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed	"&"	document member of the same patent family

Date of the actual completion of the international search 16 JUNE 1997	Date of mailing of the international search report 24 JUL 1997
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer JOSEPH R. BURWELL Telephone No. (703) 305-3800

580

Element Boundaries

582 584 586 588

	Attribute Value		Child		Document Instance	
Element	start	end	start	end	start	end
top	10	14	8	19	16	20
first	0	5	0	3	0	7
second	6	8	4	6	8	13
end	9	8	7	7	14	15

590 592 594 596 598 600 602

574

Entity Table

Index	Name	Type	Notation	Value
0	18	NDATA	1	4
1	19	NDATA	0	5
2	21	NDATA	2	7
3	22	NDATA	1	9
4	25	NDATA	0	11

572

Document Instance Table

Index	GI	Flags	First Attribute	First Child	# of Children
0	2	100	0	0	0
1	0	000	1	0	1
2	3	100	1	1	0
3	0	000	2	1	1
4	3	100	2	2	0
5	0	010	3	2	1
6	2	100	4	3	0
7	0	010	5	3	1
8	2	100	6	4	0
9	3	000	7	4	1
10	3	100	7	5	0
11	4	000	8	5	1
12	3	100	8	6	0
13	0	000	9	6	1
14	3	100	9	7	0
15	0	000	10	7	1
16	4	110	10	8	4
17	4	110	12	12	3
18	4	100	14	15	1
19	1	000	15	16	3
20	0	000	15	19	/

578

Child Table

Index	Type	Value
0	ELEMENT	0
1	ELEMENT	2
2	ELEMENT	4
3	ELEMENT	6
4	ELEMENT	8
5	ELEMENT	10
6	ELEMENT	12
7	ELEMENT	14
8	ELEMENT	1
9	ELEMENT	3
10	ELEMENT	5
11	ELEMENT	7
12	ELEMENT	9
13	ELEMENT	11
14	ELEMENT	12
15	ELEMENT	15
16	ELEMENT	16
17	ELEMENT	17
18	ELEMENT	18
19	ELEMENT	19

Attribute Value Table

Index	Value
0	18
1	19
2	21
3	8
4	22
5	10
6	25
7	26
8	27
9	29
10	24
11	23
12	23
13	28
14	28

576

FIGURE 13B

B. FIELDS SEARCHED

Electronic data bases consulted (Name of data base and where practicable terms used):

APS, ProQuest IEEE DB
search terms: multimedia, hypermedia, deliver?, tree, SGML, ODA, tagged elements, markup language, video, document, server, network, HyTime

CORRECTED VERSION*

PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 :
G06F 15/00, 17/30

A1

(11) International Publication Number: **WO 97/34240**

(43) International Publication Date: 18 September 1997 (18.09.97)

(21) International Application Number: PCT/US97/04574

(22) International Filing Date: 17 March 1997 (17.03.97)

(30) Priority Data:

60/013,505	15 March 1996 (15.03.96)	US
08/792,371	3 February 1997 (03.02.97)	US

(71) Applicant: UNIVERSITY OF MASSACHUSETTS [US/US]; One University Avenue, Lowell, MA 01854 (US).

(72) Inventors: BUFORD, John, F.; 85 Butman Road, Lowell, MA 01852 (US). RUTLEDGE, John, L.; 54 Woodbury Lane, Acton, MA 01720 (US).

(74) Agent: LAND, John; Fish & Richardson P.C., Suite 1400, 4225 Executive Square, La Jolla, CA 92037 (US).

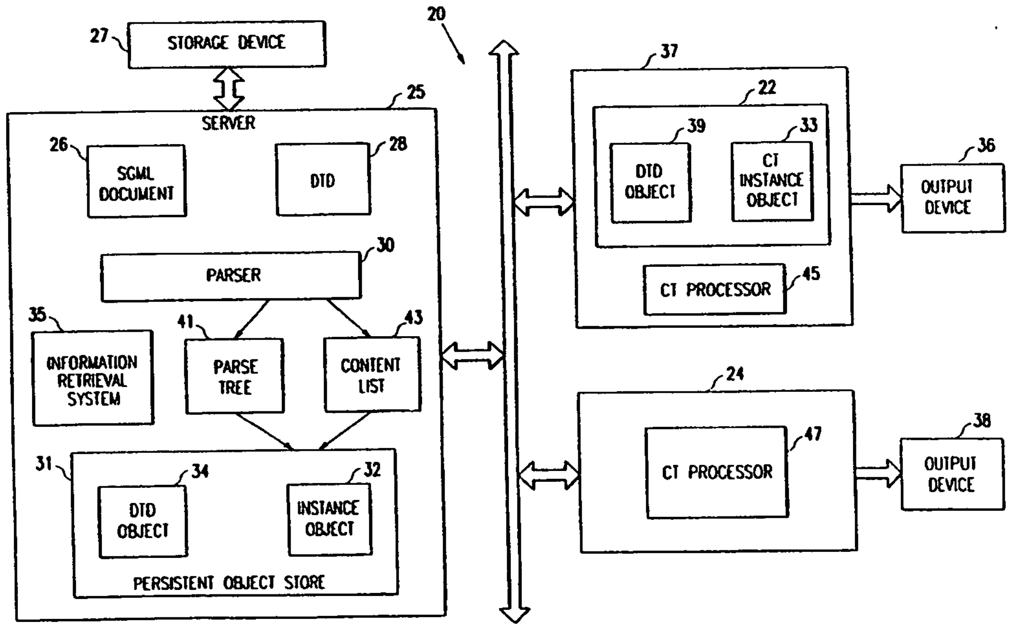
(81) Designated States: AU, CA, JP, European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: COMPACT TREE FOR STORAGE AND RETRIEVAL OF STRUCTURED HYPERMEDIA DOCUMENTS

(57) Abstract

A compact tree representation is used during the electronic storage, transmission, and presentation of a structured hypermedia document (25) in a networked computer. All text portions of the documents are pre-processed by a document parser (30) and the resulting document structure is stored in compact and compressed form in a persistent object storage (31) while the document content (32) is available in a compressed and indexable form consistent with full text retrieval systems. During document delivery, the compact representation of the document (33, 39) is retrieved from the persistent object storage and transferred to a client computer (22), which reconstructs the document and presents it to a user (36). Any arbitrary structured document type can be stored and delivered this way. The CT representation can be partitioned dynamically or by preprocessing to allow the client to retrieve parts of the document incrementally, for non-linear or temporally ordered access. The CT representation can be used for group collaboration applications, including document sharing and document authoring applications.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AM	Armenia
AT	Austria
AU	Australia
BB	Barbados
BE	Belgium
BF	Burkina Faso
BG	Bulgaria
BJ	Benin
BR	Brazil
BY	Belarus
CA	Canada
CF	Central African Republic
CG	Congo
CH	Switzerland
CI	Côte d'Ivoire
CM	Cameroon
CN	China
CS	Czechoslovakia
CZ	Czech Republic
DE	Germany
DK	Denmark
EE	Estonia
ES	Spain
FI	Finland
FR	France
GA	Gabon
GB	United Kingdom
GE	Georgia
GN	Guinea
GR	Greece
HU	Hungary
IE	Ireland
IT	Italy
JP	Japan
KE	Kenya
KG	Kyrgystan
KP	Democratic People's Republic of Korea
KR	Republic of Korea
KZ	Kazakhstan
LI	Liechtenstein
LK	Sri Lanka
LR	Liberia
LT	Lithuania
LU	Luxembourg
LV	Latvia
MC	Monaco
MD	Republic of Moldova
MG	Madagascar
ML	Mali
MN	Mongolia
MR	Mauritania
MW	Malawi
MX	Mexico
NE	Niger
NL	Netherlands
NO	Norway
NZ	New Zealand
PL	Poland
PT	Portugal
RO	Romania
RU	Russian Federation
SD	Sudan
SE	Sweden
SG	Singapore
SI	Slovenia
SK	Slovakia
SN	Senegal
SZ	Swaziland
TD	Chad
TG	Togo
TJ	Tajikistan
TT	Trinidad and Tobago
UA	Ukraine
UG	Uganda
US	United States of America
UZ	Uzbekistan
VN	Viet Nam

COMPACT TREE FOR STORAGE AND RETRIEVAL OF STRUCTURED HYPERMEDIA DOCUMENTS

BACKGROUND OF THE INVENTION

1. *Field of the Invention*

This invention relates to electronic data storage, and more particularly to storage, retrieval, transmission, and presentation of structured hypermedia documents in distributed computing systems.

2. *Description of Related Art*

Structured documents are a class of electronic information storage in which the text content of a document includes embedded character sequences known as "markup" which identify structural elements and attributes or formatting codes for the content. The Standard Generalized Markup Language (SGML) is an example of a syntax for storing and processing structured documents (International Organization for Standardization, ISO International Standard 8879 – Standard Generalized Markup Language, 1985 Geneva, Switzerland). The HyTime standard is an example of a markup language for structured hypermedia documents (International Organization for Standardization, ISO International Standard 10744 – Hypermedia Time-Based Structuring Language (HyTime), 1992. Geneva, Switzerland). The Hypertext Markup Language (HTML), which is defined as an SGML document type definition (DTD), is a widely used markup language for hypertext documents (Berners-Lee, T., and Connolly, D., Hypertext Markup Language – 2.0, Internet Engineering Task Force RFC 1866, November 1995).

Under the SGML standard, a DTD defines the structural components that are required and allowed in a particular type of document. Each DTD begins with a declaration of the document type, *i.e.*, a statement that assigns an identifier to the particular document type (*e.g.*, a DTD defining a magazine may begin with the document type declaration "magazine"). All documents of the declared type should be structured according to the

DTD. The DTD then defines the elements, attributes, entities, and notations that may be used to compose a document of the given type. Elements are the components that create the logical structure of the document (*e.g.*, a magazine's elements are articles, which may consist of text, pictures, and graphical figures or tables). Attributes are the characteristics that each element type may take on in a document of the given type (*e.g.*, one attribute of an article is the page number on which it begins). Entities may be used to refer to long strings of text or to external files (*e.g.*, the term "Johnson article" may refer to an article in another magazine). Notations identify non-SGML components and provide instructions for using these components when presenting the document. Additional information on documents structured according to the SGML syntax may be found in *The SGML Primer*, SoftQuad, Inc., 1995.

One conventional approach by which structured documents in electronic form are delivered involves a client-server division. A client application executing on a local client processor retrieves a document in its native encoding from a server software program executing on a remote server processor (transfer). The client application then parses the document locally according to the grammar of the document structure (parsing), combines the output of the parser with local requirements regarding style attributes of individual elements (rendering), and presents the document and its content to the user interface (display). A retrieved document is transferred in bulk to the client.

Bandwidth is a scarce resource in present client-server systems, so it would be desirable to minimize the amount of data required to be transferred from a server to a client. Accordingly, a second approach by which structured documents are delivered is for the document server to transmit to the client a compressed form of the document's native encoding (compression), using conventional compression techniques such as those described in Witten, I. H., Moffat, A., and Bell, T., *Managing Gigabytes – Compressing and Indexing Documents and Images*, NY: Van Nostrand Reinhold 1994. The client retrieves the compressed document (transfer), decompresses the document to obtain the native encoding (decompression), and then parses the document locally according to the grammar of the document structure (parsing), combines the output of the parser with local requirements regarding style (rendering), and presents the document and its content to the user interface (display). However, a retrieved compressed document still is transferred in bulk to the client.

In these conventional approaches, a document authoring tool generates documents in a native encoding. Thereafter, the documents are transferred to the server and retained in the native encoding. If a document has been structured for time-dependent presentation (that is, some elements of the document are to be displayed before other elements), the internal scheduling information of the document is not used to schedule transmission of document elements from server to client. Instead, the document is transferred in bulk to the client for parsing, rendering, and time-dependent display at the client. Further, in the conventional approaches, multiple users accessing the same document for simultaneous collaboration each must retrieve a new copy of the entire document each time the original document is modified by another user.

Accordingly, it would be useful if the efficiency of the present client-server system of structured document access could be improved. The present invention provides a system and method that provides such improved efficiency.

SUMMARY OF THE INVENTION

The present invention provides a system and method of storage, retrieval, transmission, and presentation of structured hypermedia documents in client-server distributed computing systems. In the preferred embodiment, a network server maintains a persistent storage of an arbitrary number of preprocessed structured documents. Each structured document is processed and parsed once at the network server, and the result is stored in compact tree (CT) form in a persistent object store. The CT form of the document then is delivered to requesting client computers as one or more objects. When a document is edited at a client computer, the CT form or the edit operations can be transferred from the client to the server.

Since the document is retrieved in pre-parsed and pre-processed form, the client computer need not have a document parsing function. The parsing operation is done once at the server, and thereafter the resulting representation may be accessed many times by different clients. The clients do not need to parse the compact representation, because the representation retrieved by the client contains a parse tree and symbol definitions needed to render the document for presentation.

The compact tree techniques described above result in significantly smaller data transfers than traditional document transfer techniques. The actual bandwidth reductions depend upon the size and structure of the structured documents being transferred. Furthermore, transferring compact tree representations incrementally provides additional bandwidth and network performance improvements.

Since the CT representation and the original source representation of the document are essentially equivalent, the CT representation may replace the source representation, thereby saving storage space and cache memory space at the server. Similarly, since client computers generally cache documents in source format, cache memory usage in the client computers is reduced. The CT representation also preserves cache storage space at proxy servers that act as intermediate servers between the client computers and a remote server.

Since the CT representation of a structured document reduces the time needed to access the document, the network server is able to sustain more sessions in a given interval. Because client computers do not need to parse the document, the clients are able to present the document more quickly than when using traditional document transfer techniques.

Full-text and content-based retrieval are an important requirement for hypermedia document systems. The CT representation of structured documents separates the document's structural elements from its content, allowing the content to be stored and indexed by conventional full-text or content-based retrieval systems, while the structural portions can be queried by a structure-query processing language such as the HyQ query language [ISO 1992].

The CT representation also allows non-linear, partial-retrieval (incremental), and progressive access to documents. Since a document is stored on the server as a compact parse tree, client computers may access a subsection of the tree without retrieving and processing the entire tree. This speeds client access, particularly for large documents, since the compact tree can be partitioned into sub-trees deliverable in an arbitrary order, including non-linear and temporal ordering. Even though incremental access is useful in supporting non-linear and temporally-ordered access to portions of a document, conventional document transfer processes do not permit incremental, per document file access. Processing time also is reduced because the syntactical validation performed by the parser is performed only once at the server, rather than each time the document is accessed by a client.

In collaborative situations, multiple client computers may simultaneously view, and may even modify, the same document. To maintain document consistency among all clients during collaboration, changes made to the document at each client are propagated to the server and to the other clients. The object-oriented storage model of the document permits edit transaction objects to be incorporated into the incremental delivery of the document.

Advantages of the invention may include one or more of the following:

- A computer network may be simplified by eliminating the need for document parsers in client computers requesting access to structured documents.
- A structured document may be parsed once by a network server and then accessed indefinitely by multiple client computers.
- Network storage and bandwidth requirements may be reduced by transferring a compact representation of a structured document instead of transferring the entire document itself.
- Network performance may be improved by reducing the time required to access a structured document.
- Conventional information retrieval systems may be used to store and index the content of a structured document.
- Modifications to a structured document may be sent to a client user who is viewing the document without retransmitting the entire document.

The details of the preferred embodiment of the present invention are set forth in the accompanying drawings and the description below. Once the details of the invention are known, numerous additional advantages, innovations, and changes will become obvious to one skilled in the art.

BRIEF DESCRIPTION OF THE DRAWINGS

FIGURE 1 is a functional block diagram of a computer network that stores and transmits structured documents in a compact tree representation.

FIGURES 2A and 2B are flow diagrams of a network server's processing of a structured document.

FIGURE 3 is a flow diagram of a network client's processing of a compact tree representation of a structured document.

FIGURE 4 is a structural diagram of the elements of a document type definition (DTD) object for a compact tree representation of structured documents.

FIGURE 5 is a schematic diagram of the elements of an instance object for a compact tree representation of a structured document.

FIGURES 6A and 6B are block diagrams illustrating the storage and transmission format of a DTD object.

FIGURE 7 is a block diagram illustrating the storage and transmission format of the instance object.

FIGURES 8A and 8B are block diagrams of a computer network that stores and transmits structured documents in a partitioned compact tree representation.

FIGURES 9A and 9B are block diagrams of a computer network in which modifications to a structured document are transmitted between network computers.

FIGURES 10A and 10B are a document type definition (DTD) and an example SGML document generated according to the DTD, respectively.

FIGURES 11A and 11B are tables found in compact tree representations of the DTD and SGML document of FIGURES 10A and 10B.

FIGURE 12 is another example SGML document generated according to the DTD of FIGURE 10A.

FIGURES 13A and 13B are tables found in partitioned compact tree representations of the SGML document of FIGURE 12 and the corresponding DTD.

Like reference numbers and designations in the various drawings indicate like elements.

DETAILED DESCRIPTION OF THE INVENTION

Throughout this description, the preferred embodiment and examples shown should be considered as exemplars, rather than as limitations on the present invention.

Referring to FIGURE 1 and FIGURES 2A and 2B, in a first embodiment of the present invention, a structured document 25 encoded according to a standard syntax, such as SGML [ISO 1988], may be shared among the various computers of a computer network 20. In general, a network server computer 26 will receive a request for the document 25 from at least one of a set of client computers 22 and 24 (Step 50). Upon receiving the request, the server 26 retrieves the document 25, either from an associated storage device 27 if the server 26 is a computer system that permanently maintains the document 25 (*e.g.*, the server 26 is an Internet server), or from another computer system if the server does not permanently maintain the document (*e.g.*, the server 26 is a proxy server) (Step 52). The document 25 is accompanied by a DTD file 28 that defines the syntactical elements used to convert the document 25 into a compact tree (CT) representation. The DTD file 28 allows the document to be reconstructed from the CT representation and to be presented as formatted output.

After receiving the document 25, the server 26 uses an internal parser 30 to parse through the SGML document 25, breaking the document 25 into a parse tree 41 representing the document's structural hierarchy and a content list 43 representing the data contained in the document 25 (Step 54). The server 26 then converts the parse tree 41 and the content list 43 into structure and character data (CDATA) tables, respectively (Step 56). The structure tables store the document's structural information in compressed form, and the CDATA table stores the data content of the document 25 in compressed and indexable form. These tables are described in detail below. The server 26 then stores the structure and CDATA tables as a compact tree (CT) instance object 32 in a persistent data store 31, such as an object-oriented database, an object-oriented file system, or an object-relational database (Step 58). The instance object 32 is structured such that related structural elements (*i.e.*, the document's elements and their corresponding sub-elements) are located together in the compact tree, which allows the clients 22 and 24 and the server 26 to access the instance object, and therefore the structured document, incrementally, as described below. If the server 26 includes an information retrieval system 35 (Step 60), the server 26 passes the CDATA portion of the instance object 32 to the information retrieval system 35 for storage (Step 62). The server 26 also compresses the DTD file 28 into a compact DTD object 34 and stores it in the persistent object store 31 (Step 66), if the server 26 has not already done so while previously processing a document of the same type (Step 64). The server 26 uses standard compression techniques to create the instance object 32 and the DTD object 34.

Instead of delivering the entire SGML document 25 to the requesting client 22 or 24, the server 26 delivers only a copy 33 of the CT instance object 32 and, if necessary, a copy 39 of the CT DTD object 34. If the client 22 or 24 requests only a portion of the document (Step 68), the server 26 sends only the sub-objects of the CT instance object 32 relating to the requested portion (Step 70). Otherwise, the server 26 transmits the entire CT instance object to the requesting client 22 or 24 (Step 72). Furthermore, the server 26 transmits the DTD object to the client 22 or 24 (Step 76) only when the client specifically requests the DTD object 34 (Step 74). Since the DTD may be any SGML-conforming definition, the client generally will need to retrieve a copy 39 of the DTD object 34 with the document instance object 32. However, if the client 22 or 24 can process the document instance 32 without the DTD object 34, or if the client 22 or 24 already has a copy 39 of the DTD object 34, then only a copy 33 of the document instance object 32 need be transmitted to the client 22 or 24. The server 26 then waits for the next incremental or full-document request from the client 22 and 24 (Step 78).

If a document is structured according to a syntax that the server 26 does not recognize, the server 26 may follow one of several approaches. First, the server 26 may reject the document completely and notify the requesting client that the document could not be delivered. Second, if only a portion of the document is unrecognizable, the server 26 may ignore the unrecognizable portion and transmit the rest. Third, the parser 30 may replace the unrecognizable syntax with an acceptable syntax. Fourth, the server 26 may transmit the unrecognizable syntax to the client as character data (CDATA).

Referring also to FIGURE 3, after requesting the document or a portion thereof from the server 26, the client 22 or 24 receives a copy 33 of the CT instance object 32 or of the appropriate sub-objects (Step 82) and stores it in a local storage area 37, such as cache memory or a hard disk drive (Step 84). If the client 22 or 24 has not already received the DTD object 34 in response to an earlier request (Step 86), that client requests the DTD object 34 (Step 88) and, after receiving it, stores the object in the local storage area 37 (Step 90). Using the DTD object as a guide, a CT processor 45 or 47 in the client 22 or 24 then traverses the CT instance object (Step 92) and converts that object into a formatted document (Step 94). The client 22 or 24 then presents the formatted document to an output device 36 or 38, such as a video display or a printer (Step 96).

In a second embodiment of the present invention, documents are pre-processed in the server 26 so that they are available in processed form before requested by a client. In a third embodiment of the present invention, as documents are requested by clients and processed in the server 26 as above, the processed form is stored in the server so that they are available in processed form for future requests by a client.

Referring to FIGURE 4, a CT DTD object 34 preferably is organized into several tables, each of which contains a portion of the DTD information needed for recreation (rendering) and presentation of the structured document. In the preferred embodiment, the tables in the DTD object 34 include a generic identifier (GI) table 102, a name table 104, an attribute declaration table 106, an attribute enumeration table 108, a notation table 110, an entity table 112, and a character data (CDATA) table 114. Preferably, DTD information that is needed only at parse time (*e.g.*, the content model of each generic identifier) is discarded and is not placed into the DTD object 34. The name table 104 and the CDATA table 114 preferably consist of text strings; the other tables preferably consist of constant length numerical fields. In the preferred embodiment, the name table 104, notation table 110, entity table 112, and CDATA table 114 are "common" tables that may be shared by the DTD object 34 and the instance object 32. The GI table 102, attribute declaration table 106, and attribute enumeration table 108 are DTD-specific tables.

To simplify access to the information stored in the DTD object 34, each table preferably has constant-size entries which may be identified by simple numeric indices. Client computers can find information in a table by multiplying the entry index for the desired information by the predetermined size of the entries. Variable length information, such as a list of sub-elements (or "children" elements) associated with an element, preferably is stored as consecutive entries in either the name table 104 or the CDATA table 114. As a result, all variable length information may be represented by two numeric values: one value representing the index of the first table entry containing a portion of the desired information, and the other value representing the number of entries required to store the information. Storing variable length information in constant size table entries trades space efficiency for access efficiency. One of ordinary skill in the art would recognize that the tables may be designed differently to strike a different balance between space efficiency and access efficiency.

In the preferred embodiment, the GI table 102 maintains a list of generic identifiers associated with the attributes contained in the attribute declaration table 106. The GI table 102 preferably contains at least four fields for each generic identifier. The first field 120 indicates the index of the generic identifier string stored in the name table 104. The second field 122 indicates the index of the first attribute stored in the attribute declaration table 106 for each generic identifier. As described below, all attributes associated with a generic identifier are stored consecutively in the attribute declaration table 106, with fixed attributes followed by non-fixed attributes. The third field 124 and fourth field 126 of the GI table 102 indicate the number of fixed attributes and the number of non-fixed attributes, respectively, associated with each generic identifier. If desired, other information may be stored in the GI table 102.

The name table 104 preferably stores a single copy of the text string associated with each generic identifier. Each string contained in the table 104 is represented by an index that identifies the position of the string in the table 104. The indices are represented by a binary value containing the least number of bits required to identify every generic identifier (*i.e.*, $\log_2 N$, where N is the number of strings in the name table 104). By storing all identifier strings in the name table, the DTD object 34 reduces the storage space required to represent all string identifiers.

The attribute declaration table 106 maintains a list of the attributes associated with each generic identifier, their types, and their default values. The table 106 preferably includes at least three fields: a name field 128, a type field 130, and a value field 132. The name field 128 indicates the index of the attribute name stored in the name table 104. The type field 130 indicates whether the attribute is an identifier (ID), an IDREF, a name, an entity reference, an enumeration, or CDATA. If the attribute is an identifier (ID or IDREF), the value field 132 stores the index of the identifier string in the name table 104. If the attribute is an enumeration, the value field 132 indicates the index of a list of possible enumeration values stored in the attribute enumeration table 108. If the attribute is CDATA, the value field 132 indicates the index of the default value in the CDATA table 114. Because this information is stored in the attribute declaration table 106, the structured document and the CT instance object may contain a list of attributes that differ from their default values. The attribute declaration table 106 also eliminates the need to store information about fixed attributes in the document instance object.

The attribute enumeration table 108 maintains a list of possible enumeration values for each attribute declared in the attribute declaration table 106. The attribute enumeration table preferably contains at least two fields. An NameTable Entry field 133 stores the index of a first enumeration value in the name table 104. A No. of Entries field 134 stores the number of possible enumeration values. These enumeration values are stored consecutively in the name table 104, starting at the index stored in the NameTable Entry field 133. Optionally, an index field 135 is also provided to reference multiple entries in the attribute enumeration table 108. (Indeed, throughout these examples, all of the index columns are shown for clarity but are not explicitly part of the CT representation).

The notation table 110 preferably includes at least three fields: a name field 140, a type field 142, and a value field 144. The name field 140 stores an index to the notation identifier string stored in the name table 104 for each notation. The type field 142 contains a flag (preferably a single bit) for each notation that indicates whether the notation is a system or public notation. The value field stores an index to the corresponding notation value in the CDATA table 114.

The entity table 112 preferably consists of at least four fields: a name field 146, a type field 148, a notation field 150, and a data field 152. The name field 146 stores an index to the entity identifier string stored in the name table 104 for each entity. The type field 148 indicates whether the entity is notation data (NDATA), character data (CDATA), or specific character data (SDATA), each of which is known in the art. If the entity type is NDATA, the notation field 150 stores an index to the corresponding notation string in the notation table 110; otherwise, the notation field 150 is empty (or ignored). The data field 152 stores an index to a corresponding entity definition in the CDATA table 114.

The CDATA table 114 preferably stores unparsed character data, such as notation values, external entity values, and attribute values. Like the name table, the CDATA table 114 stores each variable length string in consecutive constant-length entries that may be indexed easily. After the DTD object 34 is fully constructed, the CDATA table 114 is compressed using a conventional text compression technique.

Referring to FIGURE 5, a CT instance object 32 also is composed of several tables, each of which contains a portion of the information required to reconstruct the document instance. In the preferred embodiment, the tables specific to the instance object 32 include at least an instance table 160, an attribute value table 162, and a child table 164, each of which is described below. These tables describe the document-specific element hierarchy and identify any attributes that are not set to the corresponding default values. The instance object 32 also may use the four common tables in the DTD object (*i.e.*, CDATA, name, notation, and entity). The instance object 32 preferably will include an entity table (*i.e.*, will use the common entity table) only when new external entities are used in the document instance. Likewise, the instance object 32 will include a notation table, GI table, or CDATA table only when new notations, identifiers, or CDATA, respectively, are used in the document instance. The index values for the instance object elements in the common tables are determined by treating each table as a continuation of the corresponding table in the DTD object.

The instance table 160 indicates the elements occurring in the document instance. The instance table 160 preferably includes at least five information fields for each element: a generic identifier (GI) field 166, a flag field 168, a first attribute field 170, a first child field 172, and a number-of-children field 174. The GI field 166 stores an index to a corresponding generic identifier in the GI table 102 of the DTD object. The flag field 168 contains a binary bit vector in which each bit represents a non-fixed attribute in the corresponding element. If the bit corresponding to a particular attribute is set, that attribute does not have the default value, but rather its value is determined by the value found in the first attribute field 170. The first attribute field 170 stores the index of the first non-default attribute value stored in the attribute value table 162 for the corresponding element. The attributes for which the corresponding bits in the flag field 168 are set are stored consecutively in the attribute value table 162, beginning at the index stored in the first attribute field 170. All attributes which are set to the corresponding default values are represented by cleared bits in the flag field 168 and are omitted in the attribute value table 162. The bits in the flag field bit vector and the attribute values in the attribute value table 162 appear in the same order that the non-fixed attributes for the corresponding generic identifier appear in the attribute declaration table 106. The first child field 172 stores the index of the corresponding element's first child element (or sub-element) in the child table 164. The number-of-children field 174 indicates how many children sub-elements the corresponding element has. All sub-elements of an element are stored consecutively and in proper hierarchical order in the child table 164.

The attribute value table 162 stores at least a value for every attribute that is not set to the corresponding default value. Each entry in this table consists of one value, an index to the identifier string in the name table 104 representing the default attribute value. The attribute name and attribute type need not be stored in this table because this information is stored in the attribute declaration table in the DTD object.

The child table 164 preserves the hierarchical structure of the elements occurring in the document instance. Each entry in the child table 164 is represented by at least two information fields: a child field 176 containing a single bit indicating whether the entry is itself an element or is a pseudo element, and a value field 178 storing an index to the instance table 160 if the entry is an element or an index to the CDATA table 114 if the entry is a pseudo element. A child element contains everything located between two start and end tags found within the parent element, while the corresponding pseudo elements contain the content data located before the start tag and after the end tag, respectively.

Each field in the tables of the instance object 32 preferably contains the smallest number of bits possible to present the required information. For example, the GI field 166 of the instance table 160 should contain only enough binary bits to represent the highest index in the GI table 102 of the DTD object. Similarly, the attribute value table 162 and the child table 164 are large enough only to index the respective tables in the DTD object. In the preferred embodiment, the bit vectors in the flag field 168 of the instance table 160 must include one bit for each non-fixed attribute in the element type having the most non-fixed attributes. The number-of-children field 174 must be large enough only to indicate the number of sub-elements in the element having the most sub-elements. Each child table entry must have enough bits to store the highest index value for the CDATA table 114 and the instance table 160, plus an additional bit to indicate the child type. Each entry in the attribute value table 162 must be large enough to store the highest index in the name table 104, CDATA table 114, entity table 112, or attribute enumeration table 108, whichever is larger. The entries in the attribute value table 162 may have more bits than the value field 132 of the attribute declaration table 106 because the document instance may have added additional information to the name table 104 and the CDATA table 114.

The last entry 180 in the instance table 160 is unique in that it provides special information required by client computers to recreate the document instance. This entry preferably has at least one sub-element. Additional elements are pseudo elements containing any SGML processing instructions defined in the DTD. SGML processing instructions, which are known in the art, provide system specific information that must be used to reconstruct the document instance and therefore must be included in the instance object. The SGML processing instructions are followed by the first child that is a true element (as opposed to a pseudo element), which indicates the top level element of the document. Any remaining children of the last instance table entry 180 may be used by the server to pass optional parameters, such as PI entity definitions, that help the server and the client optimize or customize delivery of the document instance.

The entries in the document instance table 160 are ordered according to a recursive, depth-first search of the parse tree. As a result, entries representing components of the same element occupy consecutive locations in the name table 104, entity table 112, CDATA table 114, instance table 160, attribute value table 162, and child table 164. Also, the entries representing a child element preferably appear before the entries of subsequent children of the same parent element. Furthermore, the entries representing an element's children preferably immediately precede the element's own entries in the document instance table 160.

Referring to FIGURES 6A and 6B, the server stores and transmits the DTD object in a format understood by the client computers. The preferred format of the DTD-specific portion 200 of the DTD object is shown in FIGURE 6A. The server first stores a fixed length integer field 202 representing the number of enumerations contained in the attribute enumeration table 108 (FIGURE 4). The server then stores a bit-packed version of the attribute enumeration table 108 in a variable length field 206. Following the attribute enumeration information is a fixed length integer field 208 indicating the number of entries in the attribute declaration table 106 (FIGURE 4) and a variable length field 210 containing a bit-packed version of the attribute declaration table 106. The server then stores a fixed length field 212 indicating the number of entries in the generic identifier (GI) table 102 (FIGURE 4), followed by a variable length field 214 containing a bit-packed version of the GI table 102.

The preferred format of the common portion 220 of the DTD object (*i.e.*, the portion that may contain document instance information as well as DTD information) is shown in FIGURE 6B. The first two fields are a fixed length integer field 222 indicating the number of bytes in the compressed CDATA table 114 (FIGURE 4) and a variable length field 224 containing the compressed CDATA table 114. The next two fields are a fixed length integer field 226 indicating the number of entries in the name table 104 (FIGURE 4) and a variable length field 228 containing the strings of the name table 104. These fields are followed by a fixed length integer field 230 indicating the number of entries in the notation table 110 (FIGURE 4) and a variable length field 232 containing a bit-packed version of the notation table 110. The last two fields of the common portion 220 of the DTD object are a fixed length integer field 234 indicating the number of entries in the entity table 112 (FIGURE 4) and a variable length field 236 containing a bit-packed version of the entity table 112.

Referring to FIGURE 7, the server also stores and transmits the instance object 32 in a format understood by the client computers. The server preferably first stores a fixed length integer field 242 indicating the number of entries in the attribute value table 162 (FIGURE 5), followed by a variable length field 244 containing a bit-packed version of the attribute value table 162. The next two fields include a fixed length integer field 246 indicating the number of entries in the child table 164 (FIGURE 5) and a fixed length integer field 248 indicating the number of entries in the instance table 160 (FIGURE 5).

These fixed length fields are followed by two variable length fields, the first field 250 containing a bit-packed version of the child table 164, and the second field 252 containing a bit-packed version of the instance table 160. As discussed above, the last entry in the instance table 160 identifies the DTD object corresponding to the instance object 32 and provides instructions for reconstructing the document instance. The client computers automatically retrieve the last entry in the instance table 160 to begin the reconstruction process.

Referring to FIGURES 8A and 8B, the server 26 may create a CT instance object 32 in a manner that allows incremental delivery of the instance object 32 in either linear or non-linear order. Incremental delivery permits a client to retrieve and present portions of a document that are needed immediately for viewing regardless of where they occur in the document. For example, the client may need to begin its presentation of a document with a hyperlink that occurs in the middle of the document. Incremental delivery also permits a progressive-style display of a document, such as displaying all of the document's main headings before displaying its subheadings and body. Incremental access also allows the client to retrieve additional portions of a document only as the user attempts to view them, so that the server must transfer only those portions of the document that are needed by the user. This provides more optimal use of system resources when, for example, the ;user hyperlinks to a new document before entirely viewing the current document. Incremental delivery also allows the client to access portions of a temporally organized document, so that parts of a document can be retrieved in a specific time order. Incremental transfer is important for hypertext access in which the user is likely to browse through documents without viewing their entire contents.

As shown in FIGURES 8A and 8B, the server 26 creates an instance object 32 ready for incremental delivery by effectively dividing the associated compact tree into sub-trees. The server 26 divides the compact tree by partitioning the tables in the CT instance object 32. Because all of the information about an element and its children is stored as consecutive entries in the instance object tables, the instance table may be partitioned easily. To do so, the parser 30 generates an index 260 of element boundaries that indicates the first and last entries in each instance object table containing information about each element and its corresponding sub-elements. The index 260 of element boundaries preferably consists of at least three sub-indices: a first sub-index 262 that indicates the index bounds of information stored in the attribute value table 162 (FIGURE 5) for each element and its corresponding sub-elements; a second sub-index 264 that indicates the index bounds of child information stored in the child table 164 (FIGURE 5) for each element; and a third sub-index 266 that indicates the index bounds of entries in the document instance table 160 (FIGURE 5) for each element and its sub-elements. Alternatively, the server 26 may divide the document into a uniform set of equally-sized partitions without regard to corresponding elements and sub-elements. For a temporally scheduled document, the server may partition the document instance object 32 into sub-trees that preserve the time order in which the portions of the document must be presented to a client.

When a client requests incremental delivery of a document, a document delivery engine 49 in the server 26 uses the index 260 of element boundaries to determine which entries from the three instance object tables (attribute value, child, and instance) must be sent to the requesting client. Instead of sending each of the three tables in its entirety along with a fixed length integer indicating the size of the table, the document delivery engine 49 delivers the requested range of entries along with a pair of fixed length integers, one of which indicates the index of the first entry in the delivered table fragment, and the other of which indicates the total number of entries in the delivered table fragment. When the first sub-tree of the instance object is sent to the client, a third integer is transmitted by the server 26 to indicate the total size of the corresponding table, which allows the client to reserve enough memory space to receive all remaining fragments of the three tables, if necessary. The first two integers (*i.e.*, index of the first entry and the number of entries in the table fragment) are used by the client to place the corresponding table fragment in the proper position in the client's copies of the tables.

In general, the name table 104, notation table 110, and entity table 112 (*i.e.*, the common tables which may be shared by the DTD and instance objects) cannot be partitioned because the entries in these tables may be shared among sub-trees. However, the invention does not exclude the possibility that the CDATA table can be partitioned. Hence, there is the possibility that during increment transfer mode, the client computer may not receive the first CDATA partition that contains the desired DTD name (*i.e.*, the client may get some other partition first if, in some cases, the CDATA table is being partitioned). To deal with this case, a convention is used such that, during incremental transfer mode, the server will send the DTD name to the client right before the table boundaries are sent; this only needs to be done for the first partition.

When a client requests incremental delivery of an instance object that shares these tables with the DTD object, the server 26 may select between two alternative approaches. In the preferred embodiment, a single-bit flag stored in a flag register 270 is associated with each table to indicate which of the two approaches the server should use. If the flag bit is set, the table is small enough that the server 26 may send the entire table with each sub-tree. If the flag bit is cleared, the table is too large to be sent with each sub-tree, so the server 26 must select only those table entries that are needed to decode the particular sub-tree. In the latter situation, the server 26 transmits the numeric index of each entry selected from the table. The server 26 sets or clears the flag bits based upon the values of the integers associated with each table indicating the table size.

When the common tables are too large to send with each sub-tree, the server 26 may determine which entries to send to the client in one of two ways. First, the server 26, when it parses a document instance 25, may create and store a record indicating which entries in the common tables correspond to each partition in the DTD object 34. Second, when the server 26 creates a partition to deliver to the client, the server 26 may parse the portion of the original document 25 corresponding to the partition to determine which elements of the name table 104, notation table 110, and entity table are used in the partition. One of ordinary skill will recognize that the first alternative is preferred in a network that is more sensitive to increased processing overhead in the server during document delivery, and that the second alternative is preferred in a network that is more sensitive to increased storage overhead in the server.

Unlike the other three common tables, the CDATA table 114 can be partitioned since each piece of content data belongs to a unique element of the CT instance object 32. To support incremental delivery of documents, the server 26 independently compresses each entry in the CDATA table 114 and separately delivers each entry with the corresponding instance object sub-tree when incremental access is requested. This division of the information in the CDATA table should correspond to the partitioning of the document into sub-trees for depth first ordering.

To further improve incremental retrieval, the top elements of the document tree may be placed in a separate partition to which other sub-trees are attached. In large documents, the lower-level sub-trees also may be divided in a similar fashion. Sub-tree division is best performed in a DTD-specific way in order to use knowledge of the document structure for optimization.

In response to a client's request for incremental delivery of a document, the server transfers all of the entries for the highest-level sub-tree. The client then processes the high level structure of the document and requests only the sub-trees it requires. The client may use conventional addressing mechanisms such as SGML IDREF or HyTime location addressing forms to identify any element instance within a document. When the client returns a request for a specific element, the server uses the address of the requested element to create an instance object sub-tree containing information for the requested element and its corresponding sub-elements.

Progressive views of the document can be provided to the client by passing consecutive partitions of the same root sub-tree incrementally, but with successively increasing depth.

Alternatively, a root sub-tree having N levels can be delivered as a partition to the client, and the client can present the sub-tree progressively to the N levels.

Referring again to FIGURE 1, the server 26 may be a proxy server acting as an intermediate server between a conventional server and its client computers, including clients located behind a "firewall" and mobile clients. The proxy server may serve as a bridge to improve system security or system performance. If a remote server passes a native encoding document 25 to the proxy sever, the parser 30 in the proxy server will convert the document 25 into the compact tree (CT) format and then forward the CT document to the client.

Referring to FIGURES 9A and 9B, client computers 302 and 304 may engage in simultaneous collaborative author-mode access to a document stored as a CT instance object 306. At the same time, each client 302 and 304 may hold similar copies 308 and 310 of the CT instance object 306 for simultaneous viewing and modification. When one client 302 modifies its copy 308 of the instance object by deleting, inserting, appending, or replacing information, a document editor 312 in the client computer 302 creates an edit object 320 that stores the modifications. The edit object 320 preferably includes one fixed length integer field 322 containing a time-stamp for the modifications and another fixed length integer field 324 indicating the address of the instance object sub-tree affected by the modifications. The edit object 320 also preferably includes a table 326 having at least two fields, the first 328 of which indicates the index of each element in the instance object that was modified, and the second 330 of which identifies the modification that was made.

The modifying client 302 sends the edit object 320 to the server 300, which in turn sends the edit object 320 to the other client 304. A document editor 314 in the server 300 then uses the edit object to modify the instance object 306 accordingly. Likewise, a document editor 316 in the other client 304 uses the edit object 320 to modify its copy 310 of the instance object accordingly. Edit objects may be used with incremental delivery.

Example

FIGURES 10A and 10B show a document type definition (DTD) 352 and an SGML document 350 generated according to the DTD 352, respectively. The DTD 352 defines a slide show presentation with time dependent information. According to the DTD 352, each slide show presentation is made up of one or more slide elements 354. Each slide 354 in turn consists of one or more items 356 (or child elements), each item being either an image element 358 or an audio element 360. Each slide 354 also has two attributes: a "name" attribute 362 that uniquely identifies the corresponding slide, and a "next" attribute 364 that indicates the name of the next slide in the presentation.

Each item element 356 (image or audio) has two attributes providing temporal information about the presentation of the element. A "start" attribute 366 indicates an amount of time to delay presentation of the item after the corresponding slide presentation has begun. When the "start" attribute 366 is set to the default value 368 of zero, the image or audio element is presented precisely when the presentation of the slide begins. A "units" attribute 370 indicates whether the "start" attribute 366 is measured in seconds 372, minutes 374, or hours 376. According to the DTD 352, "seconds" is the default value 378 of the "units" attribute 370.

Image elements 358 and audio elements 360 each include a data attribute 380 and 381, respectively, that is an SGML external entity, such as an image file or an audio file. The DTD 352 defines a notation 382 for image elements ("gif' represents the ".gif' format for image data) and a notation 384 for audio elements ("ulaw" represents the ".ulaw" format for audio data). Both notations 382 and 384 are defined as "SYSTEM" type notations. Image elements 358 also include an "x" attribute 386 and a "y" attribute 388, which define the x-y location of the image in the slide presentation space. Both the "x" and "y" attributes 386 and 388 have default values 390 and 391 of zero.

In FIGURE 10B, the document instance 350 declares that the "slideshow" document is defined by the document type definition "slideshow.dtd" 464, shown in FIGURE 10A.

The document instance 350 also declares a notation and five entities not declared in the DTD 352. The declared notation 464 ("postscript") is assigned the Adobe PostScript 3.0 format 466 ("PS-Adobe-3.0"). The first two entities, an "audio-1" entity 468 and an "audio-2" entity 470, are "NDATA" entities associated with two audio data files, and "audio-1.ulaw" file 472 and an "audio-2.ulaw" file 474, respectively. The other three entities, an "image-1" entity 476, an "image-2" entity 478, and an "image-3" entity 480, are "NDATA" entities associated with three image data files, an "image-1.gif" file 482, "doc.ps" 486, and an "end.gif" file 484, respectively.

After the document type and entity declarations, the document instance 350 defines the highest-level element in the document hierarchy, the "slideshow" element 488, which has two sub-elements, a first "slide" 490 and a second "slide" 492. The first "slide" element 490 has two attributes, a "name" attribute 494 that indicates the name of the first "slide" element 490 ("first") and a "next" attribute 496 that indicates the name of the second slide element. Because no slide follows the second "slide" element 492, the second "slide" element 492 has only a "name" attribute 498 indicating the name of the second slide 492 ("end").

The "first" slide 490 consists of four items elements, two audio elements 500 and 506, and two image elements 502 and 504. The first audio and image elements 500 and 502, which include the "audio-1" and "image-1" entities, are displayed simultaneously with the beginning of the first slide 490 since no delay period is specified. The second image element 504, which includes of the "image-2" entity, is displayed two seconds after the first image element 502 is displayed (*i.e.*, the "start" attribute 507 for the second image 504 has a value of "2", and the "units" attribute 508 has a value of "seconds"). The second audio element 506, which includes of the "audio-2" entity, begins 130 seconds after the beginning of the "first" slide 490 (*i.e.*, the "start" attribute 510 for the second audio element has a value of "130").

The "end" slide 492, which is presented after the "first" slide 490, consists of a single image element 512. This image element 512 includes the "image-3" entity and is displayed simultaneously with the beginning of the "end" slide presentation.

FIGURES 11A and 11B show the tables that make up the compact trees (CT) for the DTD 352 and document instance 350. The CDATA table 392, the name table 394, and the notation table 396 are common tables containing data representing both the DTD 352 and the document instance 350. The other tables shown in FIGURE 11A contain data representing only the DTD 352, and the tables in FIGURE 11B contain data representing only the document instance 350.

The name table 394 of FIGURE 11A includes 18 entries corresponding to the DTD 352 of FIGURE 10A and 8 entries corresponding to the document instance 350 of FIGURE 10B. The first entry (index 0) in the name table 394 is a null string, which is associated with objects that do not have a name (*e.g.*, the last entry in the document instance table). The next 17 entries are entered according to a depth-first recursive pass through the DTD 352. Therefore, elements appearing at lower levels of the DTD hierarchy appear first in the name table 394, while elements appearing at higher levels of the DTD hierarchy appear later in the name table 394. For example, the "audio" element 360 and the "image" element 358 are the lowest level elements in the DTD 352, so the corresponding entries in the name table 394 (index 1 and index 3, respectively) appear before the entries representing "item" element 356 (index 6), "slide" element 354 (index 12), and "slideshow" element 353 (index 15). Likewise, the attributes associated with each element immediately follow the element name in the name table 394, unless a similarly named attribute appears higher in the table. For example, the entry representing the "data" attribute 381 of the "audio" element 360 immediately follows the name table entry for the "audio" element 360, but is not duplicated below the entry for the "image" element 358, which also includes a "data" attribute 380. Likewise, entries corresponding to the "x" attribute 386 and "y" attribute 388 of "image" element 358 immediately follow the entry for the "image" element 358. The last two entries (indexes 16 and 17) in the DTD portion of the name table 394 are associated with the "gif" and "ulaw" notations 382 and 384 declared in the DTD 352.

To reconstruct the original SGML document 350 from the tables of the compact tree, the computer requesting the document first reads the last entry (index 13) in the document instance table 450 to identify the corresponding document type definition and the location of the top level element (the "slideshow" element 488) in the document instance table 450. Because the last entry in the document instance table 450 does not refer to an element of the document, no generic identifier or attribute is associated with the entry. Therefore, the corresponding "GI" and "first attribute" column entries are not examined. The client computer obtains the name of the required DTD from the last entry in the CDATA table 392 (index 12). The last entry in the document instance table 450 has at least one corresponding entry in the child table 444, one of which identifies the location of the highest level element in the document instance table 450. In this case, the last entry in the document instance table 450 includes exactly one child element, which is stored as the last entry in the child table 444 (index 12). This child element indicates the location (index 12) of the highest level element ("slideshow") of the compact tree in the document instance table 450.

The computer then reads the document instance table entries for the highest level element to gather information about that element and its children elements. The "GI" field 452 contains a value of "1", indicating that the corresponding generic identifier information is located in the second position (index 1) of the GI declaration table 416. The "identifier" field 418 in the GI declaration table contains a value of "15", indicating that the name of the highest level element ("slideshow") is located in the sixteenth entry (index 15) in the name table 394. The "first attribute" field 420 in the GI declaration table contains a value ("8") greater than the highest index of the attribute declaration table 408, which indicates that the "slideshow" element has no corresponding attributes. The requesting computer then returns to the document instance table and finds that because the "slideshow" element has no corresponding attributes, the "flags" field 454 has no bits set and the "first attribute" field 456 contains an invalid index value ("10"). The "first child" field 458 and the "number of children" field 460 indicate that the "slideshow" element has two sub-elements at the eleventh and twelfth positions in the child table 444. The "value" field 448 of the child table 444 indicates that information for the first of these sub-elements is located in the ninth position (index 8) of the document instance table 450 and that information for the second sub-element is located in the twelfth position (index 11) of the document instance table 450.

Reading the information contained in the ninth entry of the document instance table 450, the computer learns that the generic identifier for the sub-element is contained in the fifth position (index 4) of the GI declaration table 416. This entry in the GI declaration table 416 points the computer to the thirteenth position (index 12) of the name table 394, which indicates that the sub-element is a "slide" element. The attribute fields 420, 422, and 424 of the GI declaration table 416 indicate that each slide element has two associated unfixed attributes that are identified by the seventh and eighth entries (indices 6 and 7) in the "attribute" declaration table 408. The first attribute of each "slide" element is an IDREF-type attribute, the name ("next") of which is contained in the fifteenth entry (index 14) in the name table 394. The "next" attribute has no default value, as indicated by the null string contained in the first entry (index 0) of the CDATA table 392. The second attribute of the "slide" element is an ID-type element, the name of which ("name") is contained in the fourteenth entry (index 13) in the name table 394. The "name" attribute also has a null default value.

After identifying the "name" and "next" attributes associated with the "slide" element, the computer returns to the ninth entry of the document instance table 450 and reads the values of the bits in the corresponding entry of the "flags" field 454. Because each of the first two bits is set, the computer knows it must retrieve the actual values of the "name" and "next" attributes from the attribute value table 440. The "first attribute" field 456 directs the computer to the seventh and eighth positions (indices 6 and 7) of the attribute value table 440, which in turn refers the computer to the twenty-fourth and twenty-fifth positions (indices 23 and 24) of the name table 394. From the name table, the computer learns that the "name" of the first slide element is "first" (index 24) and that the "next" slide element is the "end" slide element (index 23). The "first child" and "number of children" fields 458 and 460 indicate that the "first" slide element has four sub-elements, which are listed sequentially beginning at the fifth position (index 4) of the child table 444. The computer then accesses the value field 448 of the child table 444 to learn that information for the four sub-elements of the "first" slide element is contained in the second, fourth, sixth, and eighth entries (indices 1, 3, 5, and 7) of the document instance table.

The computer next accesses information for the first sub-element of the "first" slide. The corresponding entry in the "GI" field 452 of the document instance table 450 indicates that the generic identifier is identified in the first entry (index 0") in the GI declaration table 416. The "identifier" field 418 of the GI declaration table 416 in turn directs the computer to the seventh entry (index 6) of the name table 394, which identifies the first sub-element as an "item"element. The GI declaration table 416 then directs the computer to the fifth and sixth entries (indices 4 and 5) in the attribute declaration table 408 for information about the two unfixed attributes associated with the "item" element. The first attribute is a CDATA attribute, the name of which ("start") is located in the twelfth position (index 11) of the name table 394. The "start" attribute has a default value of "0" as indicated by the "default" field 414 of the attribute declaration 408 and the second entry (index 1) of the CDATA table 392. The second attribute of the "item" element is the "units" attribute, as indicated by the "name" field 410 of the attribute declaration table 408 and the eighth entry (index 7) in the name table 394. The "units" attribute has three possible values, "seconds", "minutes", and "hours", which are referenced by the enumeration table 404 as the ninth through eleventh positions (indices 8-10) of the name table 394. The default value of the "units" attribute is "seconds".

After identifying the attributes associated with the "item" element, the computer returns to the document instance table 450 and skips the "first attribute" field since none of the bits in the "flags" field are set. The computer then reads information from the "first child" field 458 and the "number of children" field 460 to learn that the first "item" element has one sub-element identified by the first entry (index 0) in the child table. The "value" field 448 of the child table 444 directs the computer to the first entry (index 0) in the document instance table 450 for information about this sub-element. The "GI" field 452 of the document instance table 450 indicates that the generic identifier for the sub-element is identified by the third entry (index 2) of the GI declaration table 416. The "identifier" field 418 of the GI declaration table 416 indicates that this sub-element is an "audio" element, as specified in the second position (index 1) of the name table 394. The "unfixed attribute" field 424 of the GI declaration table 416 indicates that the "audio" element has a single attribute which, according to the "first" field 420 of the GI declaration table 416, is identified by the first entry (index 0) of the attribute declaration table 408. The "name" field 410 of the attribute declaration table 408 and the third entry (index 2) of the name table 394 indicate that this attribute is a "data" attribute. This "data" attribute is an entity having a Null default value, as indicated in the "default" field 414 of the attribute declaration table 408 and the first entry (index 0) of the CDATA table 392. Because the first bit of the corresponding entry in the "flags" field 454 of the document instance table 450 is set, the computer must look to the attribute value table entry (index 0) identified in the "first attribute" field 456 of the document instance table 450 to learn the value of the "data" attribute. This entry in the attribute value table 440 directs the computer to the nineteenth entry (index 18) in the name table 394 (*i.e.*, the first name table entry corresponding to the document instance), which indicates that the name of the "data" attribute is "audio-1". Also, because the attribute is an entity, the computer accesses the first entry (index 0) in the entity table to determine which entity is associated with the "audio-1" attribute. The "type" field 434 of the entity table 430 indicates that the entity is of the type "NDATA", and the "notation" field 436 indicates that the entity associated with the "audio-1" attribute is defined in the second entry (index 2) of the notation table 396. The "name" field of the notation table 396 indicates that the entity has the "ulaw" notation, as listed in the eighteenth entry (index 17) of the name table 394. The notation table 396 also indicates that the "audio-1" attribute is a SYSTEM type object, as indicated in the "value" field 402 of the notation table 396. The "value" field 438 of the entity table 430 then directs the computer to the fifth entry (index 4) of the CDATA table 392 for the name of the entity associated with the "audio-1" attribute. The CDATA table 392 specifies that the "audio-1.ulaw" data file is associated with the "audio-1" attribute of the first "audio" element. The computer returns to the first entry (index 0) of the document instance table 450 and learns that the first "audio" element has no sub-elements.

The computer then moves to the fourth entry (index 3) of the document instance table 450, which represents the second sub-element of the first "slide" element. Like the first sub-element of the "slide" element, the second sub-element is an "item" element having two unfixed attributes, a "start" attribute and a "units" attribute. Because none of the bits in the corresponding entry in the "flags" field 454 of the document instance table 450 is set, both the "start" and the "units" attributes are set to the default values, which means that the second item element begins simultaneously with the first item element. The "first child" field 458 and "number of children" field 460 indicate that the second "item" element also has a single sub-element, which is identified by the second entry (index 1) of the child table 444. Information for the sub-element is contained in the third entry (index 2) of the document instance table 450.

The "GI" field 452 of the document instance table 450, the "identifier" field 418 of the GI declaration table 416, and the fourth entry (index 3) of the name table 394 indicate that the only sub-element of the second "item" element is an "image" element. The "unfixed attribute" field 424 of the GI declaration table 416 indicates that the "image" element has three unfixed attributes, which are listed consecutively beginning with the second entry (index 1) of the attribute declaration table 408. The first attribute of the "image" element is a "data" entity having a null default value. The other two attributes are the "x" and "y" values that indicate the "x-y" position of the "image" element in the presentation space. These attributes are identified in the "name" field 410 of the third and fourth entries (indices 2 and 3) in the attribute declaration table 408 and the fifth and sixth entries (indices 4 and 5) in the name table 394. The "default" field 414 of the attribute declaration table 408 and the second entry (index 1) in the CDATA table indicate that both the "x" and "y" attributes have a default value of "0". The "flags" field 454 indicates that the "x" and "y" attributes of the first "image" element have the default values, while the value of the "data" attribute of the first "image" element is specified by the second entry (index 1) of the attribute value table. The "value" field 442 of the attribute value table 440 and the twentieth entry (index 19) of the name table 394 identify "image-1" as the name of this "data" attribute. The second entry (index 1) of the entity table and the first entry (index 0) in the notation table 396 identify the "image-1" attribute as a SYSTEM type object of the ".gif" file type. The "image-1" attribute is associated with the "image-1.gif" data file, as indicated by the sixth entry (index 5) of the CDATA table 392.

The computer then retrieves information about the other two sub-elements of the first "slide" element from the sixth and eighth entries (indices 5 and 7) in the document instance table 450. The computer learns that each of the sub-elements is a "item" element having a single sub-element. The first of these "item" elements includes an "image" sub-element that is associated with the "PS-Adobe-3.0" data file and that has a presentation delay of two seconds. The second of these "item" elements includes an "audio" sub-element that is associated with the "audio-2.ulaw" data file and that has a presentation delay of 130 seconds.

After the computer displays the second "audio" sub-element of the first "slide" element, it moves to the second "slide" element in the "slideshow" presentation. As discussed above, the second "slide" element is represented by information contained in the twelfth entry (index 11) of the document instance table 450. Because the second "slide" element is the last element of the "slideshow" presentation, the first bit of the "flags" field 454 and the document instance table 450 is cleared and the second bit is set, indicating that the "next" attribute of the "slide" element has the null default value, while the "name" attribute has the value ("end") identified by the last entry (index 9) in the attribute value table 440. The second "slide" element has a single "item" sub-element, which in turn has a single "image" sub-element associated with the "end.gif" data file. The third "image" element is associated with the "end.gif" data file and is displayed at the default "x-y" location (0,0) and with the default presentation delay of zero seconds. Once the second "slide" element has been presented, the requesting computer has fully reconstructed and displayed the "slideshow" document.

The first four entries of the CDATA table 392 represent information in the DTD, so every CDATA field in one of the other tables in FIGURE 11A must contain two bits to reference an entry in the CDATA table 392. Likewise, the first 18 entries of the name table 394 represent information in the DTD, so every name field in the other tables of FIGURE 11A must contain five bits to reference an entry in the name table 394. Therefore, each of the first two entries in the notation table 396, both of which represent information in the DTD, must be eight bits in length: five bits representing the index of the notation name 398 in the name table 394; one bit indicating the notation type 400 (*i.e.*, "SYSTEM" or "PUBLIC"); and two bits representing the index of the notation value 402 in the CDATA table 392. Because the notation table 396 has two entries representing information in the DTD, only one bit is needed in the other tables of FIGURE 11A to access information in the notation table 396. No entities are defined in the DTD of FIGURE 10A, so no entity table is shown in FIGURE 11A.

The DTD of FIGURE 10A defines only a single enumeration with three possible values. Therefore, the enumeration table 404 references only three entries, beginning at the index in the name table 394 indicated by a five-bit NameTable field 406.

Each entry in the attribute declaration table 408 includes three fields: a five-bit name field 410 representing the index of the attribute name in the name table 394; a three-bit type field 412 indicating which of the possible types (name, entity, CDATA, IDREF, ID, or enumeration) each attribute takes on; and a five-bit default value field 414 representing the index of the default value in the name table 394. Because the attribute declaration table 408 has eight entries, every entry in one of the other tables of FIGURE 11A referencing the attribute declaration table 408 must contain four bits.

In the preferred embodiment, each entry in the GI declaration table 416 has 11 bits: five bits representing the generic identifier name 418 in the name table 394; four bits representing the index of the first attribute 420 in the attribute declaration table 408; and two bits representing the number of unfixed attributes 424 (no GI in the DTD has more than three possible unfixed attributes). Because the DTD of FIGURE 10A defines no fixed attributes defined for any GI, no bits are needed in the "number of fixed attributes" field 422.

In addition to the four entries representing the DTD, the CDATA table 392 also includes nine entries representing the document instance. Therefore, any CDATA fields in the document instance tables of FIGURE 11B must contain four bits to reference an entry in the CDATA table 392. The name table 394 contains eight entries representing the document instance, yielding 26 total entries in the name table 394. As a result, five-bit fields are used in the document instance tables of FIGURE 11B to reference the name table 394. The notation table 396 contains one entry representing the document instance, in addition to the two DTD entries, so two-bit fields are used in the document instance tables to reference the notation table 396. In the preferred embodiment, the entry in the notation table 396 representing the document instance uses 10 bits: five bits representing the index of the notation name 398 in the name table 394, one bit indicating the notation type 400 ("PUBLIC"), and four bits representing the index of the notation value 402 in the CDATA table 392.

Because the document instance of FIGURE 10B defines five entities, the compact tree for the document instance includes an entity table 430. Entries in the other document instance tables contain three bits to reference the five entries in the entity table 430. Each entry in the entity table 430 itself contains 13 bits: five representing the index of the entity name 432 in the name table 394; two indicating the entity type 434 (NDATA, SDATA, or CDATA); two representing the index of the entity notation 436 in the notation table 396; and four representing the index of the entity value 438 in the CDATA table 392.

The compact tree for the document instance also includes an attribute value table 440, each entry of which includes a five bit attribute value field 442 representing the index of the attribute name in the name table 394. A child table 444 includes a one-bit field 446 indicating the type of each child entry ("ELEMENT" or "PSEUDO-ELEMENT") and a four-bit field 448 representing the index of the child entry in either the CDATA table 392 or the document instance table 450, discussed below.

The document instance table 450 has 15 bits per entry. Five bits represent the index of the entry's GI 452 in the GI declaration table 416. A three-bit flag vector 454 indicates which, if any, of the three possible unfixed attributes are not set to the corresponding default value. For "slide" elements (GI table index of "4" and name table index of "12"), the first bit in the flag field 454 indicates whether the "name" attribute is set to the default value, and the second bit indicates whether the "next" attribute is set to the default value. The third bit is not used. For "item" elements (GI table index of "0" and name table index of "6"), the first bit indicates whether the "units" attribute is set to the default value, and the second bit indicates whether the "start" attribute is set to the default value. The third bit is not used. For "audio" elements (GI table index of "2" and name table index of "1"), the first bit indicates whether the "data" attribute is set to the default value, and the other two bits are not used. For " image" elements (GI table index of "3" and name table index of "3"), the first bit indicates whether the "data" attribute is set to the default value, the second and third bits indicate whether the "x" and "y" attributes, respectively, are set to the corresponding default values.

Each entry in the document instance table 450 also contains four bits representing the index for the attribute value table 440 entry holding the actual value of the first unfixed attribute 456 that is not set to the default value. Four bits represent the index of the entry's first child 458 in the child table 444. Because no element in the document instance has more than four children, three bits are used to indicate the number of children 460 for each entry in the instance table 450.

In this example, the actual DTD file uses approximately 455 bytes of storage space, while the compact tree representation of the DTD consumes only 128 bytes. Likewise, the actual document instance file requires approximately 731 bytes of storage space, while the compact tree representation of the document instance uses only 183 bytes.

Referring to FIGURE 12, a sample SGML document instance 550 generated according to the DTD 352 of FIGURE 10A includes a top-level "slideshow" element 552 and three "slide" sub-elements: a "first" slide element 554, a "second" slide element 556, and an "end" slide element 558. The simple structure of the document instance 550 lends the corresponding parse tree naturally to partitioning at each slide element, though the parse tree also could be partitioned in other ways.

Referring to FIGURES 13A and 13B, the compact tree representations of the DTD 352 and the document instance 550 include a name table 560, a notation table 562, a CDATA table 564, an enumeration table 566, an attribute declaration table 568, a GI declaration table 570, a document instance table 572, an entity table 574, an attribute value table 576, and a child table 578, all similar to those described above. The CT representation for the document instance also includes an "element boundaries" table 580, which identifies the entries in the attribute value, child, and document instance tables that correspond to each of four partitions, or sub-trees, in the CT representation of the document instance. The first partition 582 ("top") identified in the element boundaries table 580 includes only the top-level elements of the document: the "slideshow" element 552 and each of the "slide" sub-elements 554, 556, and 558. The next partition 584 ("first") in the element boundaries table 580 includes only those sub-elements associated with the "first" slide element 554. Likewise, the next partition 586 ("second") includes only those sub-elements associated with the "second" slide element 556, and the last partition 588 ("end") includes only those sub-elements associated with the "end" slide element 558.

For each sub-tree, the element boundaries table 580 includes seven information fields: an "element" field 590 identifying the sub-tree; an "attribute value start" field 592 and an "attribute value end" field 594 identifying the indices of the first and last entries, respectively, in the attribute value table 576 corresponding to the sub-tree; a "child start" field 596 and a "child end" field 598 identifying the indices of the first and last entries, respectively, in the child table 578 corresponding to the sub-tree; and a "document instance start" field 600 and a document instance end" field 602 identifying the indices of the first and last entries, respectively, in the document instance table 572 corresponding to the sub-tree.

When the requesting computer first requests partitioned delivery of the document, the CDATA table 564, the name table 560, the notation table 562, and the entity table 574 are sent in their entirety, but only the portions corresponding to the "top" sub-tree 582 are sent for the attribute value table 576 (indices 10 through 14), the child table 578 (indices 8 through 19), and the document instance table 572 (indices 16 through 20). When the requesting computer later requests a child element of one of the "slide" elements, only those entries in the attribute value, child, and document instance tables corresponding to that "slide" element are sent. For example, if a child element of the "second" slide element 556 is requested, the requesting computer receives only the seventh through ninth entries (indices 6 through 8) from the attribute value table 576, the fifth through seventh entries (indices 4 through 6) from the child table 578, and the ninth through fourteenth entries (indices 8 through 13) from the document instance table 572.

Implementation

The method of the invention may be implemented in hardware or software, or a combination of both. However, preferably, the method of the invention is implemented in computer programs executing on programmable processors each comprising a processor, a data storage system (including volatile and non-volatile memory and/or storage elements), at least one input device, and at least one output device.

Each program is preferably implemented in a high level procedural or object oriented programming language to communicate with a processor. However, the programs can be implemented in assembly or machine language, if desired. In any case, the language may be a compiled or interpreted language.

Each such computer program is preferably stored on a storage media or device (*e.g.*, ROM, flash RAM, or magnetic diskette) readable by a general or special purpose programmable processor, for configuring and operating the processor when the storage media or device is read by the processor to perform the procedures described herein. The inventive system may also be considered to be implemented as a processor-readable storage medium, configured with a computer program, where the storage medium so configured causes a processor to operate in a specific and predefined manner to perform the functions described herein.

A number of embodiments of the present invention have been described. Nevertheless, it will be understood that various modifications may be made without departing from the spirit and scope of the invention. For example, the CT instance object may have additional headers that provide information such as character set identification, sequencing of objects, time stamps, ownership, and other parameters needed for managing or optimizing a distributed hypermedia system. Also, the CT DTD object and the CT instance object may be stored and transmitted with different formats than those shown here. Furthermore, structured document syntaxes other than SGML may be supported, either by handling non-SGML documents directly or by first translating non-SGML documents into the SGML syntax. Additionally, while an implementation has been described that uses a "client request" model, other models may be used, such as a subscriber or broadcast model (client requests a subscription to information, and the server from time to time publishes that information to the client without further requests), or a server push model (a client registers for specific information, and the server from time to time sends such specific information without further request), or combinations of such models.

Accordingly, it is to be understood that the invention is not to be limited by the specific illustrated embodiments, but only by the scope of the following claims.

CLAIMS

What is claimed is:

1. A computer network comprising:

(a) a client computer that authorizes receipt of a structured document, and is capable of receiving such structured document; and

(b) a server computer that, upon receiving at least an initial receipt authorization from a client computer, retrieves the structured document, parses the document into a structural portion and a content portion, and thereafter sends at least some of the two portions to the client computer.

2. The computer network of claim 1 wherein the server also sends the client computer a document type definition corresponding to the requested document.

3. The computer network of claim 1 wherein the server compresses the structured portion of the document into a compact tree.

4. The computer network of claim 1 wherein the server compresses the content portion of the document.

5. The computer network of claim 1 further comprising a processor in the client that traverses the structured portion and the content portion of the document to reconstruct the document.

6. The computer network of claim 1 wherein the structured portion of the document is arranged to allow incremental access by the client computer.

7. The computer network of claim 1 wherein the structured portion of the document comprises a compact tree, at least one element of which comprises an associated sub-tree representing sub-elements hierarchically subordinate to the element.

8. The apparatus of claim 7 wherein the compact tree comprises a table defining the structural elements of the document.

9. The computer network of claim 1 wherein the structured document adheres to a formalized syntax.

10. The computer network of claim 9 wherein the syntax comprises SGML.

11. A computer data structure for storing structured document data in a persistent object storage of a network server computer, comprising a compact tree including a parse tree representing the structural hierarchy of the structured document, and a content list representing the data contained in the structured document.

12. A persistent object storage in a network server computer configured to store structured document data as a compact tree comprising a parse tree representing the structural hierarchy of the structured document, and a content list representing the data contained in the structured document.

13. A method of storing information representing a structured document in a persistent object storage of a network server computer, the method comprising the steps of :

(a) parsing the structured document to form a parse tree representing the structural hierarchy of the structured document and a content list representing data contained in the structured document, and

(b) storing the parse tree and the content list in the persistent object storage.

14. The method of claim 13 further comprising the step of storing in the persistent object storage a document type definition corresponding to the structured document.

15. The method of claim 13 further comprising the step of compressing the content list before storing it in the persistent object storage.

16. A method of sharing a structured document in a computer network, the method comprising the steps of :
    (a) transmitting a parse tree representing the structural hierarchy of the structured document and a content list representing data contained in the structured document from a network server computer to a network client computer,
    (b) reconstructing the structured document in the network client computer using the transmitted parse tree and content list.

17. The method of claim 16 further comprising the step of transmitting a document type definition corresponding to the structured document from the network server computer to the network client computer.

18. The method of claim 16 wherein the content list comprises compressed text data.

19. The method of claim 18 further comprising the step of expanding the compressed text data after transmitting it to the network client computer.

20. The method of claim 16 wherein the parse tree and the content list are transmitted incrementally to the network client computer.

21. The method of claim 16 wherein the parse tree is transmitted to the network client computer as partitioned sub-trees.

22. A method of displaying a structured document in a network client computer, the method comprising the steps of:

(a) receiving from a network server computer a parse tree representing the structural hierarchy of the structured document and a content list representing data contained in the structured document,

(b) reconstructing the structured document in the network client computer using the received parse tree and content list, and

(c) displaying the reconstructed structured document on a display device attached to the network client computer.

23. A computer program, residing on a computer-readable medium, comprising instructions for causing a server processor to parse a structured document data into a compact tree comprising a parse tree representing the structural hierarchy of the structured document, and a content list representing the data contained in the structured document.

24. A computer program, residing on a computer-readable medium, for representing a structured document in a persistent object storage of a network server computer, comprising instructions for causing a processor to:

(a) parse the structured document to form a parse tree representing the structural hierarchy of the structured document and a content list representing data contained in the structured document, and

(b) store the parse tree and the content list in the persistent object storage.

25. A computer program, residing on a computer-readable medium, for sharing a structured document in a computer network, comprising instructions for causing at least one processor to:

(a) transmit a parse tree representing the structural hierarchy of the structured document and a content list representing data contained in the structured document from a network server computer to a network client computer,

(b) reconstruct the structured document in the network client computer using the transmitted parse tree and content list.

26. A computer program, residing on a computer-readable medium, for displaying a structured document in a network client computer, comprising instructions for causing a processor to:

(a) receive from a network server computer a parse tree representing the structural hierarchy of the structured document and a content list representing data contained in the structured document,

(b) reconstruct the structured document in the network client computer using the received parse tree and content list, and

(c) display the reconstructed structured document on a display device attached to the network client computer.

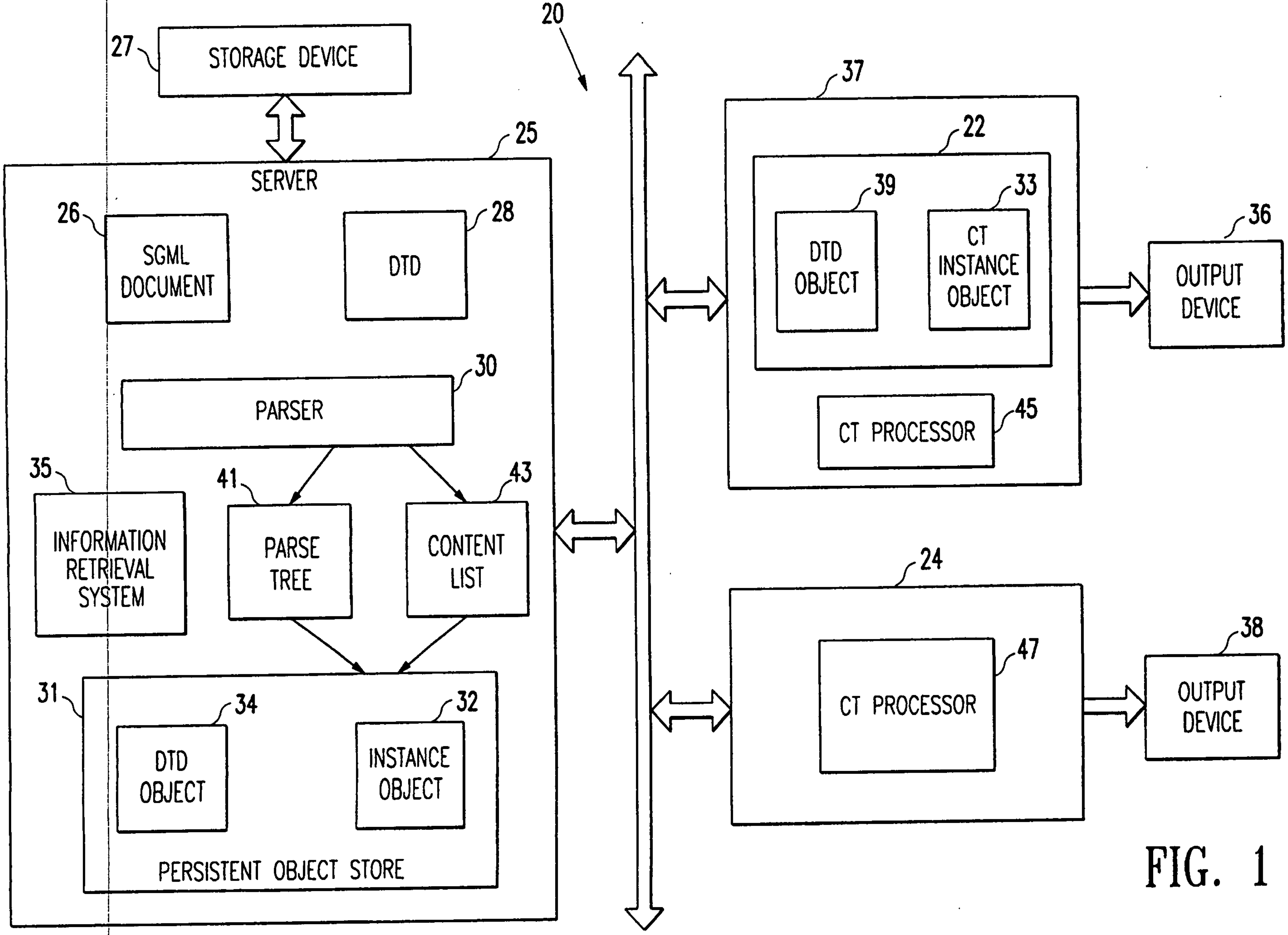

20
27
STORAGE DEVICE
25
SERVER
26
SGML DOCUMENT
DTD
28
PARSER
30
35
INFORMATION RETRIEVAL SYSTEM
41
PARSE TREE
43
CONTENT LIST
31
34
DTD OBJECT
32
INSTANCE OBJECT
PERSISTENT OBJECT STORE
37
22
39
DTD OBJECT
33
CT INSTANCE OBJECT
CT PROCESSOR
45
36
OUTPUT DEVICE
24
CT PROCESSOR
47
38
OUTPUT DEVICE

FIG. 1

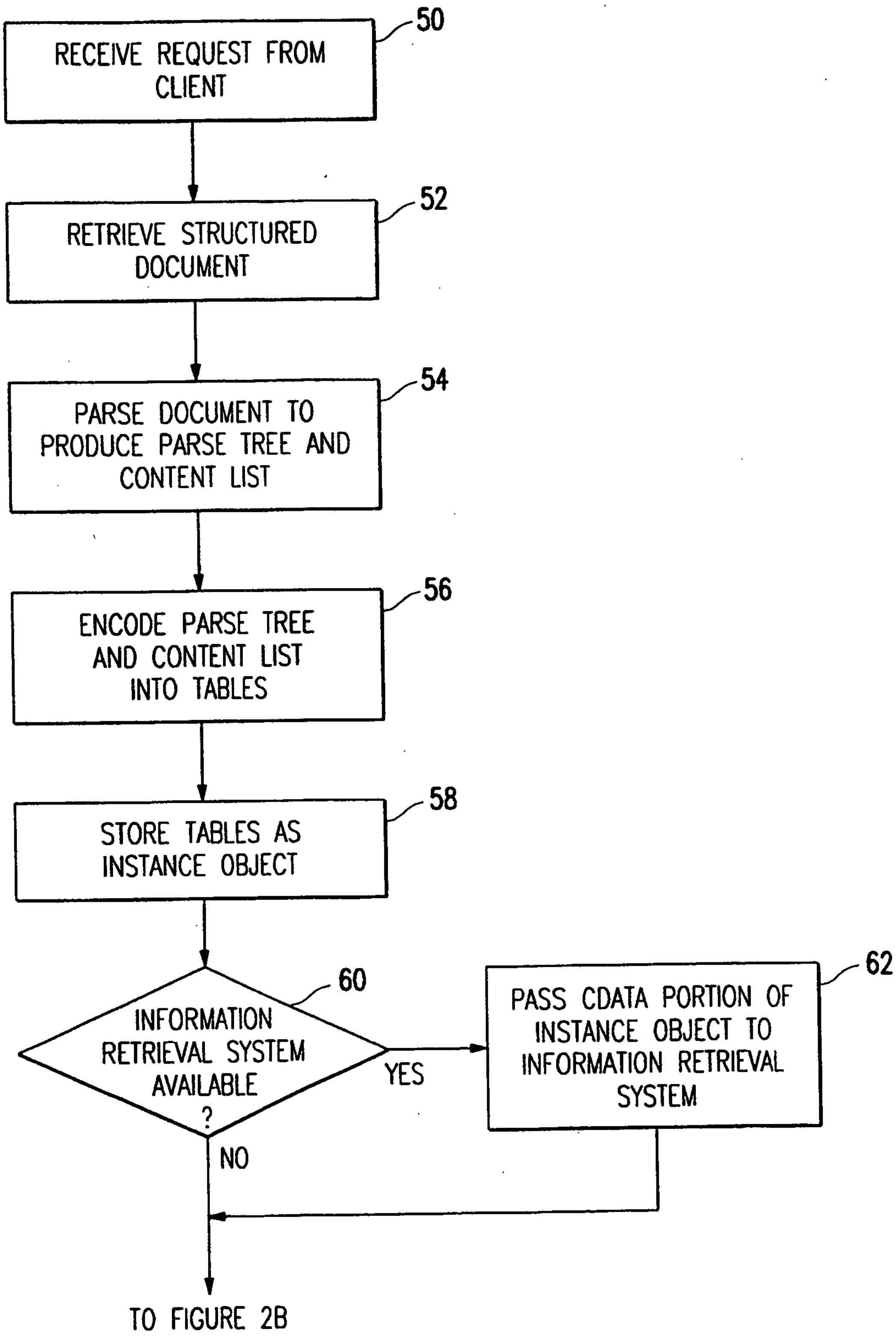

RECEIVE REQUEST FROM CLIENT
50
RETRIEVE STRUCTURED DOCUMENT
52
PARSE DOCUMENT TO PRODUCE PARSE TREE AND CONTENT LIST
54
ENCODE PARSE TREE AND CONTENT LIST INTO TABLES
56
STORE TABLES AS INSTANCE OBJECT
58
INFORMATION RETRIEVAL SYSTEM AVAILABLE ?
60
YES
PASS CDATA PORTION OF INSTANCE OBJECT TO INFORMATION RETRIEVAL SYSTEM
62
NO
TO FIGURE 2B

FIG. 2A

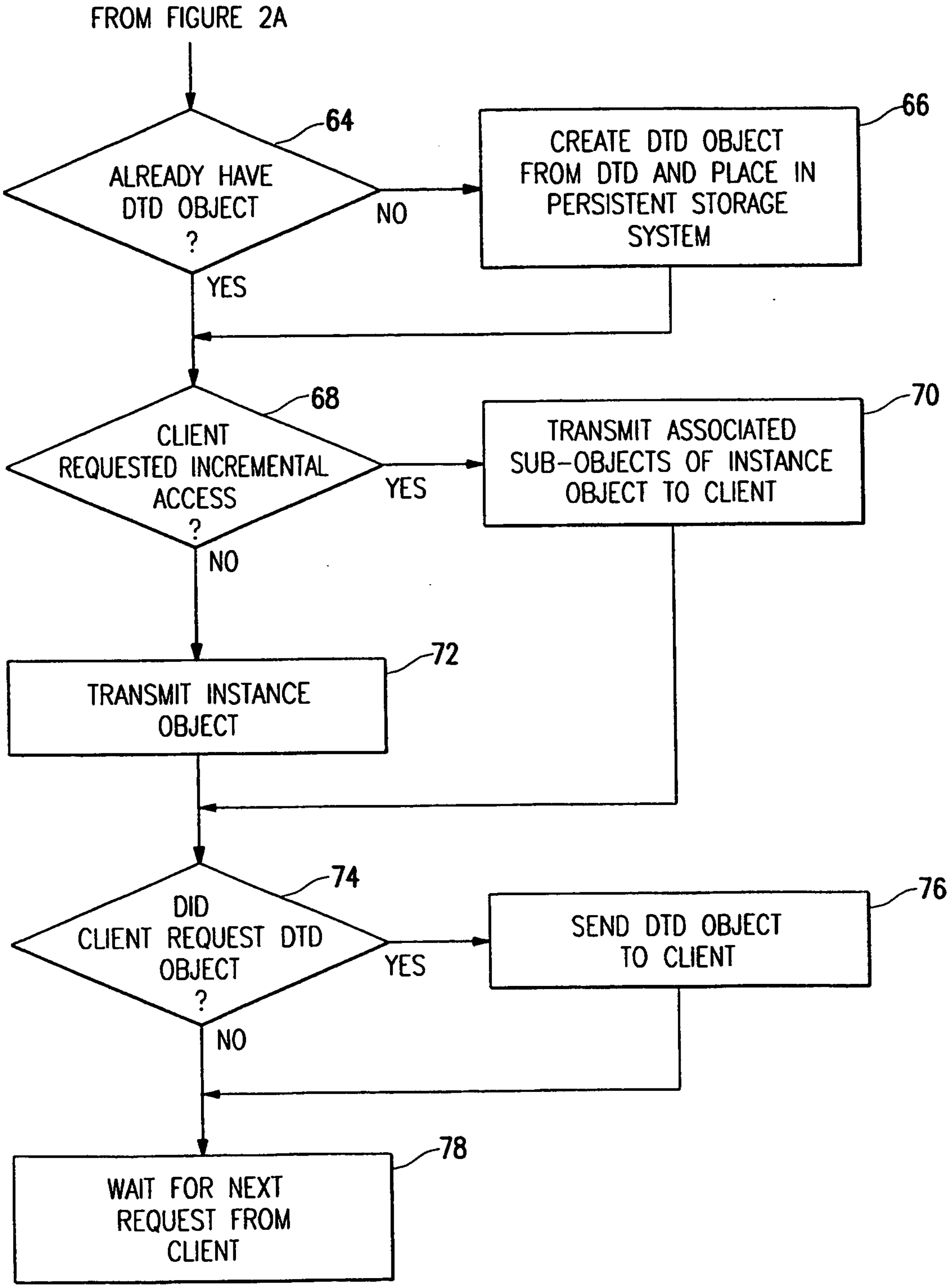

FROM FIGURE 2A
ALREADY HAVE DTD OBJECT ?
64
NO
YES
CREATE DTD OBJECT FROM DTD AND PLACE IN PERSISTENT STORAGE SYSTEM
66
CLIENT REQUESTED INCREMENTAL ACCESS ?
68
YES
NO
TRANSMIT ASSOCIATED SUB-OBJECTS OF INSTANCE OBJECT TO CLIENT
70
TRANSMIT INSTANCE OBJECT
72
DID CLIENT REQUEST DTD OBJECT ?
74
YES
NO
SEND DTD OBJECT TO CLIENT
76
WAIT FOR NEXT REQUEST FROM CLIENT
78

FIG. 2B

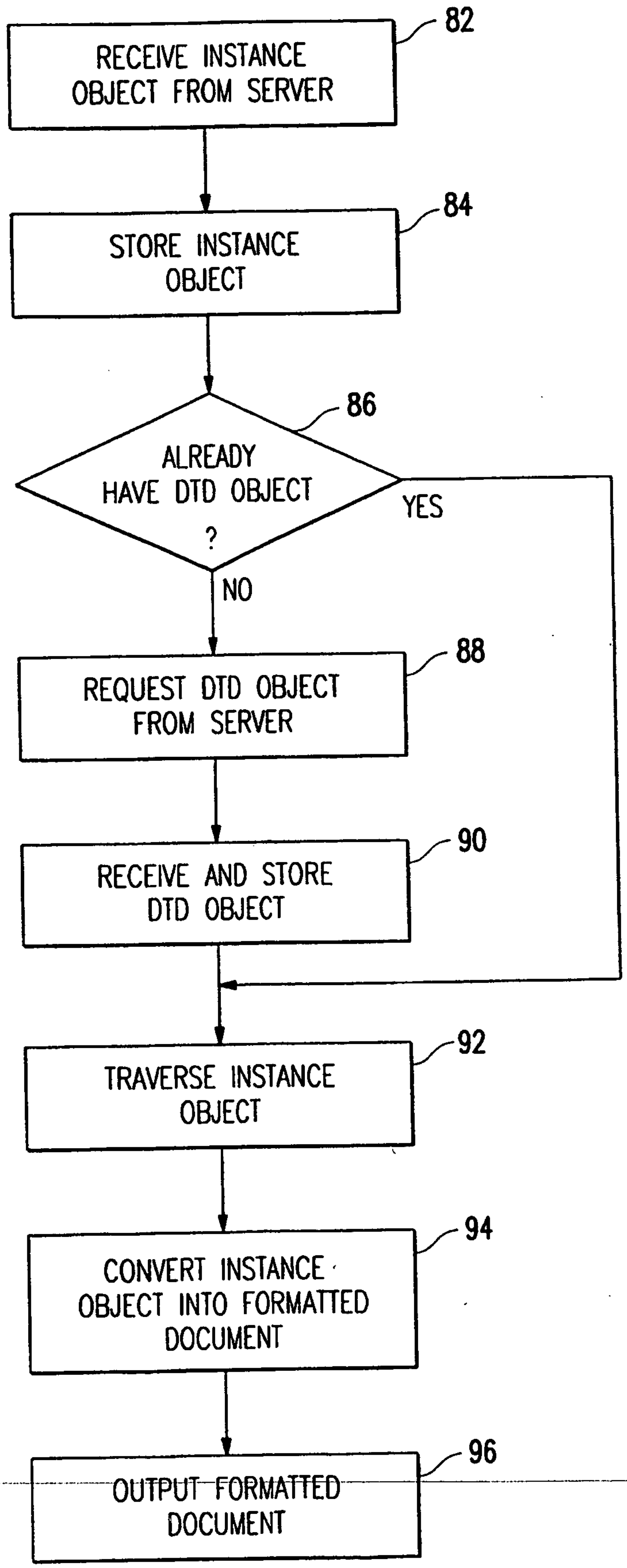

RECEIVE INSTANCE OBJECT FROM SERVER
82
STORE INSTANCE OBJECT
84
ALREADY HAVE DTD OBJECT ?
86
YES
NO
REQUEST DTD OBJECT FROM SERVER
88
RECEIVE AND STORE DTD OBJECT
90
TRAVERSE INSTANCE OBJECT
92
CONVERT INSTANCE OBJECT INTO FORMATTED DOCUMENT
94
OUTPUT FORMATTED DOCUMENT
96

FIG. 3

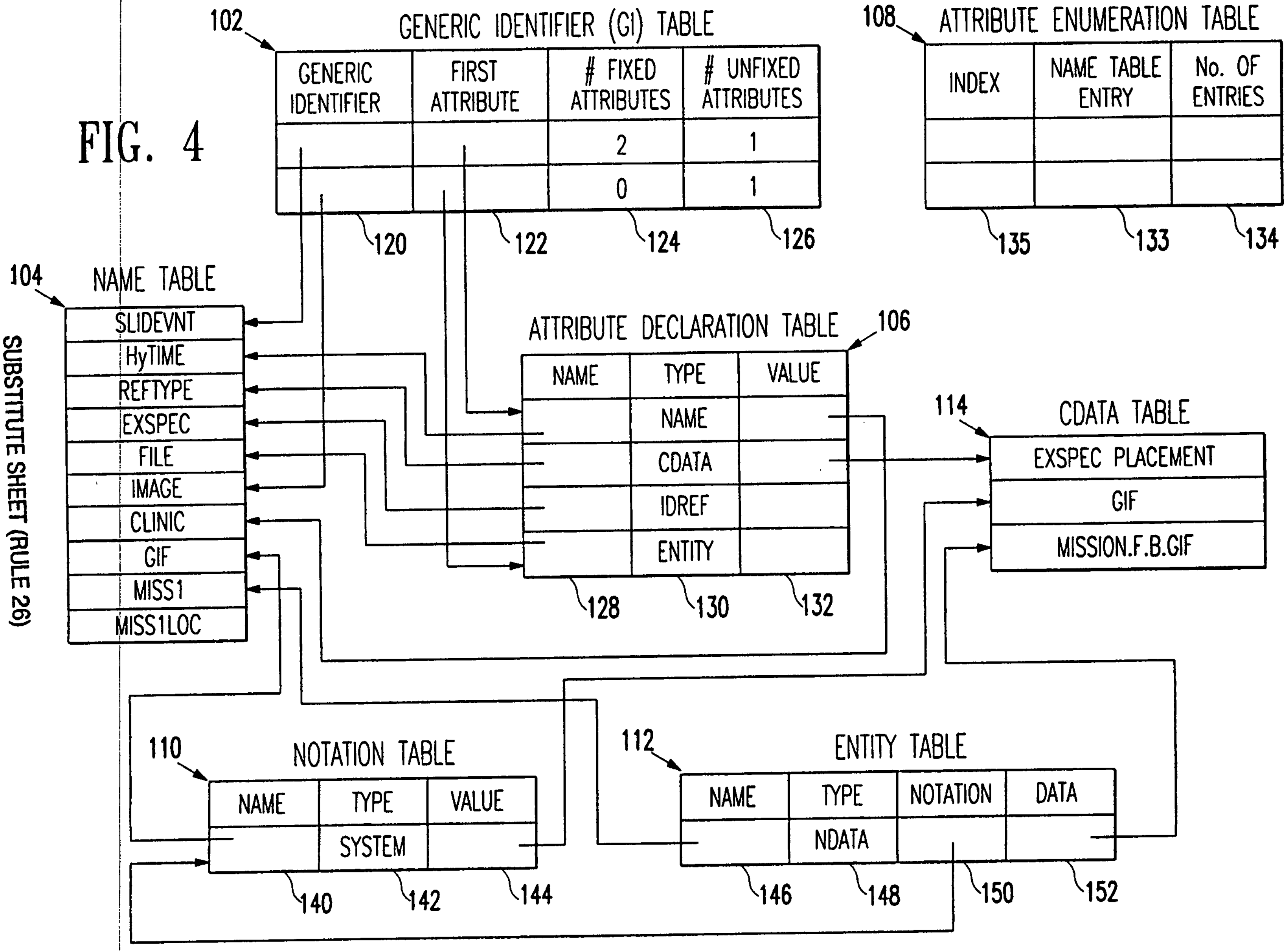

102
GENERIC IDENTIFIER (GI) TABLE
GENERIC IDENTIFIER
FIRST ATTRIBUTE
# FIXED ATTRIBUTES
# UNFIXED ATTRIBUTES
2
1
0
1
120
122
124
126
108
ATTRIBUTE ENUMERATION TABLE
INDEX
NAME TABLE ENTRY
No. OF ENTRIES
135
133
134
104
NAME TABLE
SLIDEVNT
HyTIME
REFTYPE
EXSPEC
FILE
IMAGE
CLINIC
GIF
MISS1
MISS1LOC
ATTRIBUTE DECLARATION TABLE
106
NAME
TYPE
VALUE
NAME
CDATA
IDREF
ENTITY
128
130
132
114
CDATA TABLE
EXSPEC PLACEMENT
GIF
MISSION.F.B.GIF
110
NOTATION TABLE
NAME
TYPE
VALUE
SYSTEM
140
142
144
112
ENTITY TABLE
NAME
TYPE
NOTATION
DATA
NDATA
146
148
150
152

FIG. 4

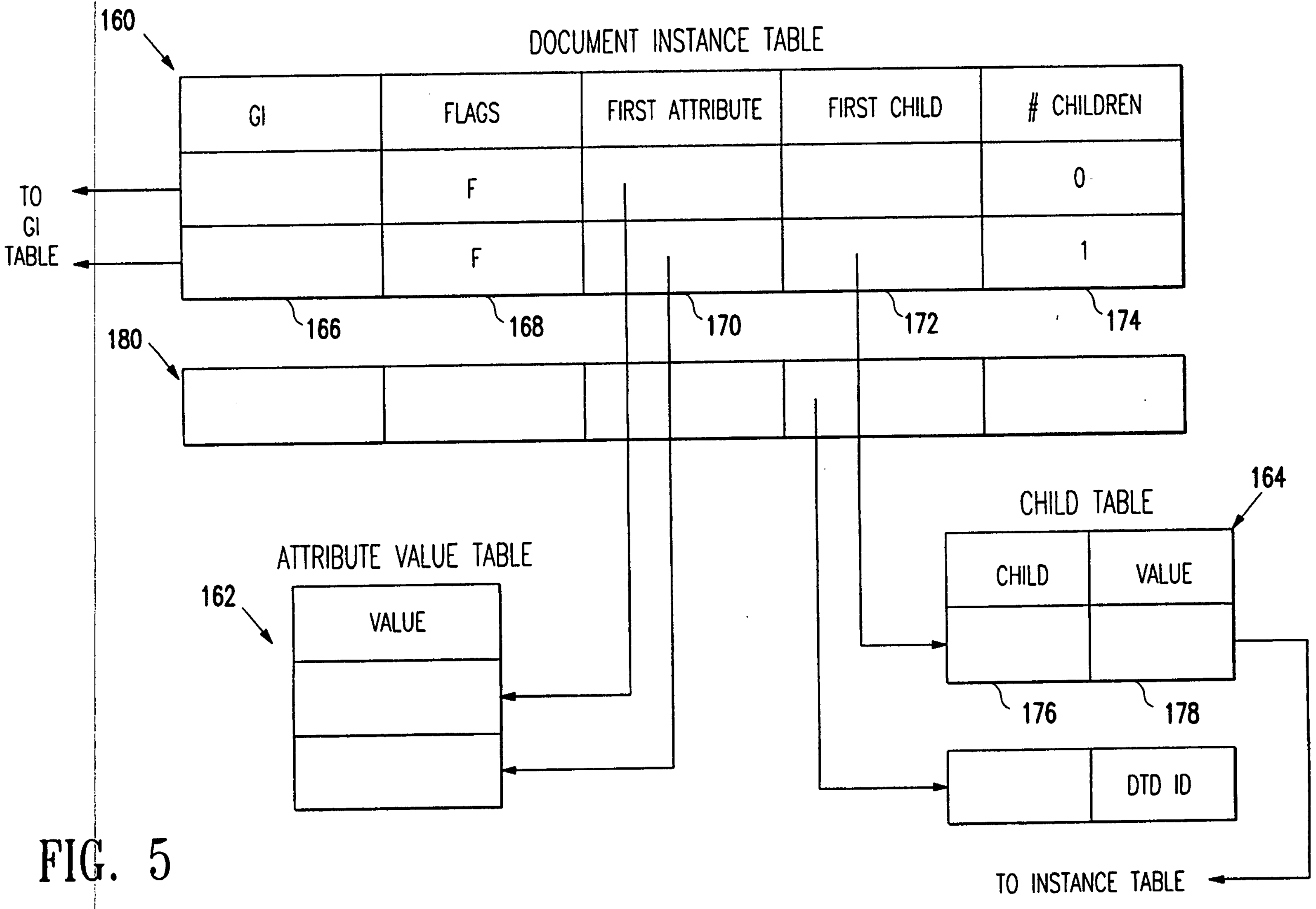

160
DOCUMENT INSTANCE TABLE
GI
FLAGS
FIRST ATTRIBUTE
FIRST CHILD
# CHILDREN
F
0
F
1
TO
GI
TABLE
166
168
170
172
174
180
ATTRIBUTE VALUE TABLE
162
VALUE
CHILD TABLE
164
CHILD
VALUE
176
178
DTD ID
TO INSTANCE TABLE

FIG. 5

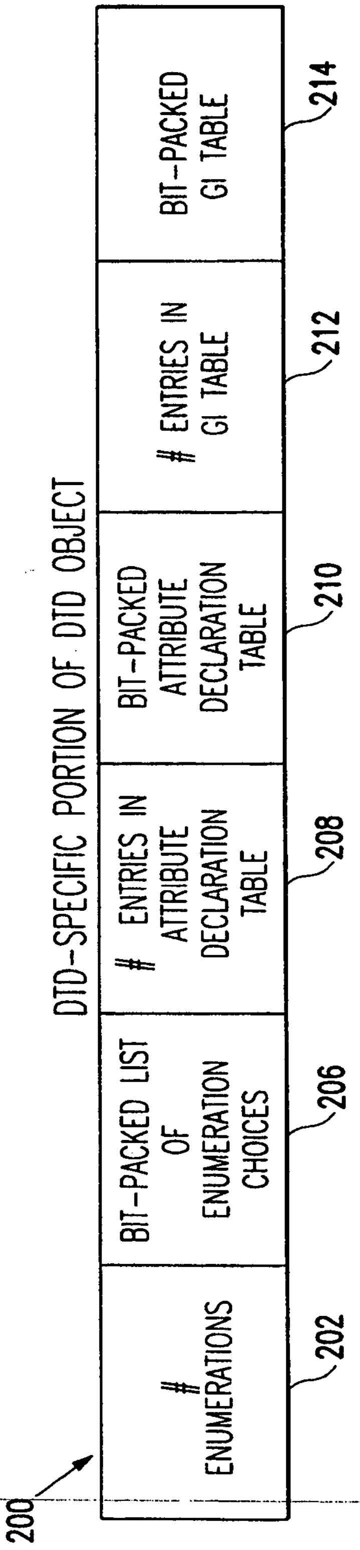

DTD-SPECIFIC PORTION OF DTD OBJECT
200
# ENUMERATIONS
202
BIT-PACKED LIST OF ENUMERATION CHOICES
206
# ENTRIES IN ATTRIBUTE DECLARATION TABLE
208
BIT-PACKED ATTRIBUTE DECLARATION TABLE
210
# ENTRIES IN GI TABLE
212
BIT-PACKED GI TABLE
214

FIG. 6A

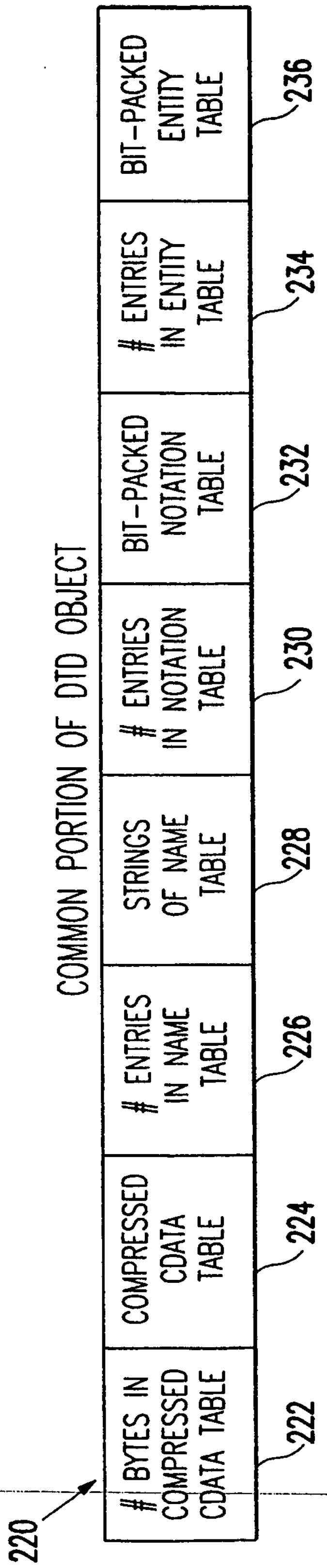

COMMON PORTION OF DTD OBJECT
220
# BYTES IN COMPRESSED CDATA TABLE
222
COMPRESSED CDATA TABLE
224
# ENTRIES IN NAME TABLE
226
STRINGS OF NAME TABLE
228
# ENTRIES IN NOTATION TABLE
230
BIT-PACKED NOTATION TABLE
232
# ENTRIES IN ENTITY TABLE
234
BIT-PACKED ENTITY TABLE
236

FIG. 6B

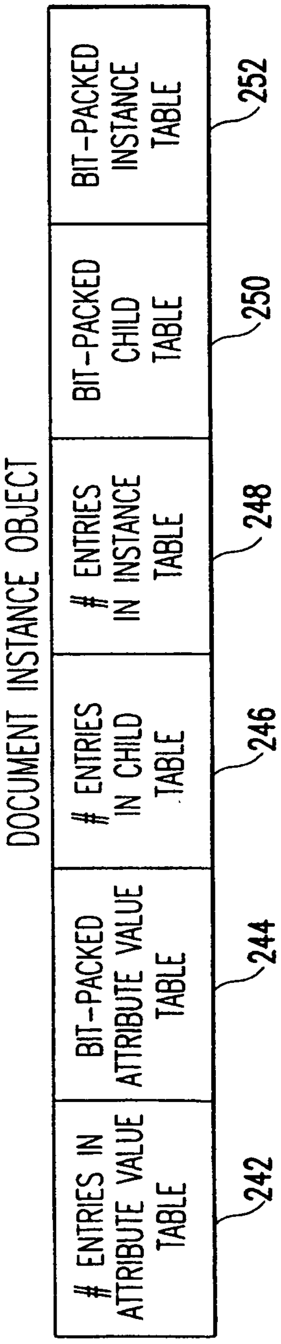

DOCUMENT INSTANCE OBJECT
# ENTRIES IN ATTRIBUTE VALUE TABLE
242
BIT-PACKED ATTRIBUTE VALUE TABLE
244
# ENTRIES IN CHILD TABLE
246
# ENTRIES IN INSTANCE TABLE
248
BIT-PACKED CHILD TABLE
250
BIT-PACKED INSTANCE TABLE
252

FIG. 7

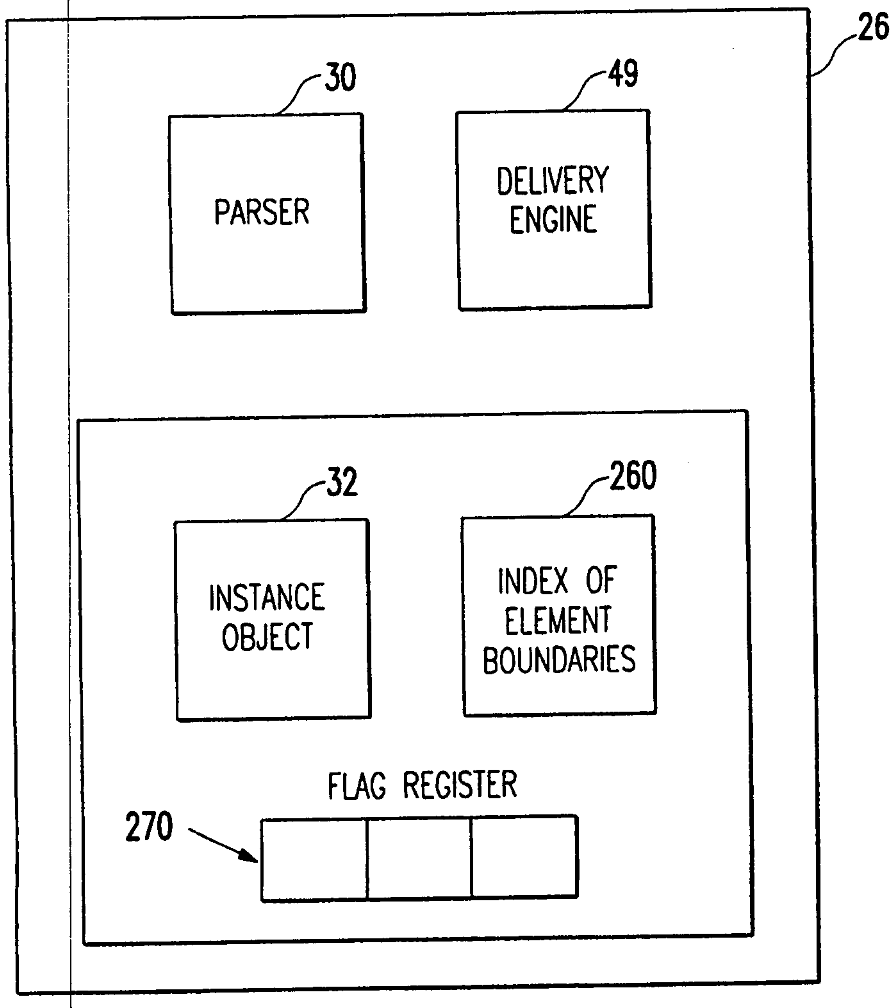

26
30
PARSER
49
DELIVERY ENGINE
32
INSTANCE OBJECT
260
INDEX OF ELEMENT BOUNDARIES
FLAG REGISTER
270

FIG. 8A

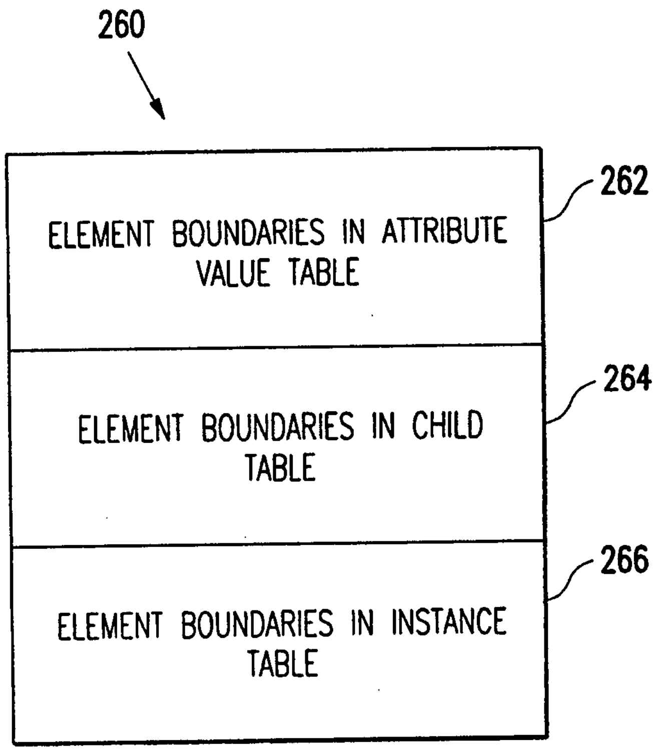

260
262
ELEMENT BOUNDARIES IN ATTRIBUTE VALUE TABLE
264
ELEMENT BOUNDARIES IN CHILD TABLE
266
ELEMENT BOUNDARIES IN INSTANCE TABLE

FIG. 8B

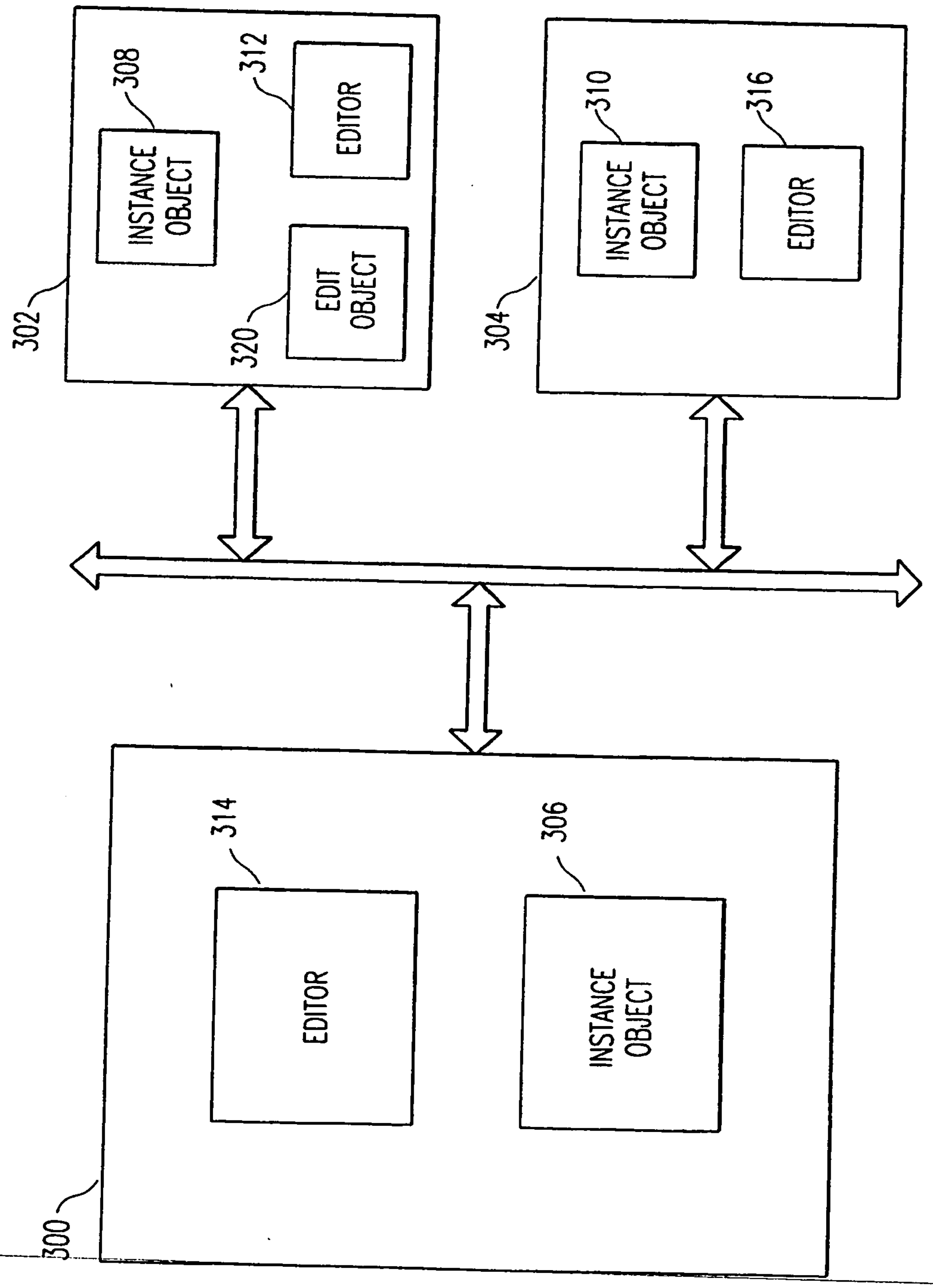

302
308
INSTANCE OBJECT
312
EDITOR
320
EDIT OBJECT
304
310
INSTANCE OBJECT
316
EDITOR
300
314
EDITOR
306
INSTANCE OBJECT

FIG. 9A